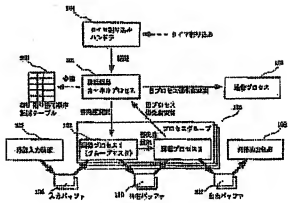
SCHEDULING METHOD FOR CYCLIC PROCESS

Publication number: JP9319597 (A)
Publication date: 1997-12-12
Inventor(s): TAKEUCHI OSAMU; NAKAHARA MASAHIKO; NAKANO TAKAHIRO; IWASAKI MASAAKI +
Applicant(s): HITACHI LTD +
Classification:
- international: ***G06F9/46; G06F9/48;*** **G06F9/46;** (IPC1-7): G06F9/46
- European:
Application number: JP19970027577 19970212
Priority number(s): JP19970027577 19970212; JP19960073673 19960328

Also published as:
JP2904483 (B2)

Abstract of **JP 9319597 (A)**
PROBLEM TO BE SOLVED: To keep constant the execution interval of respective process groups as much as possible in the case of parallelly executing the plural process groups including a process group to be cyclically executed. SOLUTION: When a group master process 102 of respective process groups 103 requests the allocation of CPU time while designating a start interval cycle and the CPU time required for each cycle, the CPU allocation time of the designated process group 103 is secured so as not to compete with the CPU allocation time of the other process group 103, and a CPU allocation order description table is prepared so as to keep the designated start interval cycle.; At the point of the time of activating any one process group 103, a kernel process 101 is activated by changing the execution priority of a process belonging to this process group 103 into the peak priority inside a system and afterwards, the peak priority is kept only for the continuously allocated CPU allocation time.



Data supplied from the ***espacenet*** database — Worldwide

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号

特開平9－319597

(43)公開日　平成９年(1997)12月12日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/46	３４０		Ｃ０６Ｆ　9/46	３４０Ｂ	
				３４０Ｅ	

審査請求　未請求　請求項の数９　ＯＬ　（全 29 頁）

(21)出願番号　特願平9－27577

(22)出願日　平成９年(1997)２月12日

(31)優先権主張番号　特願平8－73673

(32)優先日　平８(1996)３月28日

(33)優先権主張国　日本（ＪＰ）

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地

(72)発明者　竹内　理
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内

(72)発明者　中原　雅彦
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内

(72)発明者　中野　隆裕
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内

(74)代理人　弁理士　高橋　明夫

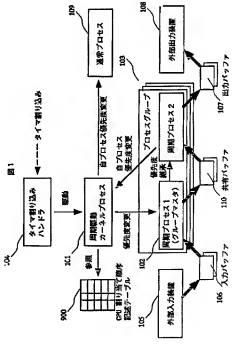
最終頁に続く

(54)【発明の名称】　周期的プロセスのスケジューリング方法

(57)【要約】

【課題】　周期的に実行されるプロセス群を含むプロセスグループが複数個並行して実行される場合の各プロセスグループの実行間隔をできるだけ一定に保つ。

【解決手段】　各プロセスグループのグループマスタプロセスが起動間隔周期と一周期ごとの必要ＣＰＵ時間を指定してＣＰＵ時間の割り当てを要求すると、指定されたプロセスグループのＣＰＵ割り当て時間が他のプロセスグループのＣＰＵ割り当て時間と競合しないように確保され、かつ指定された起動間隔周期を保つようにＣＰＵ割り当て順序記述テーブルが作成される。プロセスグループの１つを起動する時点に達したとき、カーネルプロセスは、このプロセスグループに属するプロセスの実行優先度をシステム内で最高の優先度に変更することによって起動し、その後連続して割り当てられたＣＰＵ割り当て時間分だけ最高の優先度に保つ。



【特許請求の範囲】
【請求項１】周期的に実行される少なくとも１つのプロセスを含むプロセスグループが複数存在し、これら複数のプロセスグループを並行実行させる計算機システムの周期的プロセススケジューリング方法において、
各プロセスグループごとに起動間隔周期と一周期ごとの必要ＣＰＵ時間が指定されたとき、指定されたプロセスグループのＣＰＵ割り当て時間が他のプロセスグループのＣＰＵ割り当て時間と衝突しないように確保し、かつ指定された各プロセスグループの起動間隔周期を保つように調整することを特徴とする周期的プロセススケジューリング方法。
【請求項２】請求項１記載の周期的プロセススケジューリング方法において、
タイムスロットを単位としてＣＰＵ時間を割り当てるプロセスグループの順番を登録するテーブルを作成するステップと、
起動間隔周期が短いプロセスグループから順にタイムスロットを割り当てるようにプロセスグループを選択するステップと、
選択したプロセスグループの起動間隔周期の範囲で連続して空いているタイムスロットである連続空きスロットを抽出するステップと、
必要ＣＰＵ時間以上のサイズをもつ連続空きスロット群が存在するならば、必要ＣＰＵ時間以上で最小のサイズをもつ連続空きスロットの先頭から必要ＣＰＵ時間だけの連続する空きタイムスロットを選択したプロセスグループに割り当てるステップと、
必要ＣＰＵ時間以上のサイズをもつ連続空きスロット群が存在しなければ、サイズが最大の連続空きスロットの全タイムスロットを選択したプロセスグループに割り当て、さらに必要ＣＰＵ時間から割り当てたタイムスロット分を差し引いた残り時間分について前ステップと本ステップにより該プロセスグループに割り当てるステップ、とを有することを特徴とする周期的プロセススケジューリング方法。
【請求項３】請求項１記載の周期的プロセススケジューリング方法に従ってスケジュールされたプロセスを起動する方法であって、
該プロセスグループの１つを起動する時点に達したとき、該プロセスグループに属するプロセスの実行優先度をシステム内で最高の優先度に変更することによって起動し、その後連続して割り当てられたＣＰＵ割り当て時間分だけ最高の優先度を保つことを特徴とするプロセス起動方法。
【請求項４】請求項３記載のプロセス起動方法において、
連続して割り当てられたＣＰＵ割り当て時間が経過したとき該プロセスの実行優先度をシステム内で最低の優先度に変更することを特徴とするプロセス起動方法。
【請求項５】請求項３記載のプロセス起動方法において、
連続して割り当てられた該ＣＰＵ割り当て時間が経過する前に同一プロセスグループに属する第１のプロセスから第２のプロセスへ優先度を継承するよう指示されたとき、第１プロセスの実行優先度をシステム内で最高の優先度から最低の優先度に変更し、第２プロセスの実行優先度を最高の優先度に変更することによって起動することを特徴とするプロセス起動方法。
【請求項６】請求項３記載のプロセス起動方法において、
連続して割り当てられたＣＰＵ割り当て時間が経過しかつ該必要ＣＰＵ時間を消費したとき、起動されたプロセスの実行優先度をシステム内で最高の優先度から基準の優先度に変更し、該プロセスにタイムアウトを通知することを特徴とするプロセス起動方法。
【請求項７】請求項３記載のプロセス起動方法において、
連続して割り当てられたＣＰＵ割り当て時間が経過したとき、該プロセスグループを起動する単一の制御プロセスを起動し、該制御プロセスによって次の連続するＣＰＵ割り当て時間を割り当てられたプロセスグループに属するプロセスを起動することを特徴とするプロセス起動方法。
【請求項８】請求項３記載のプロセス起動方法において、
該プロセスグループの１つを起動する時点を検出してから該プロセスグループに属するプロセスを起動するまでの処理を単一のプロセスによって実行することを特徴とするプロセス起動方法。
【請求項９】請求項３記載のプロセス起動方法において、
システム内で最高優先度をもつプロセスの実行中に、ネットワーク・パケットの到達など情報を受信すべき非同期イベントが発生したとき、直ちに最高優先度で走行中のプロセスの実行を停止し、情報の受信バッファを確保して情報を受信する準備を行った後に実行を停止したプロセスの実行を再開し、
周期的に起動されるプロセスによって受信した情報を参照して処理することを特徴とする非同期イベント処理方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明はプロセスのスケジューリング方法に係わり、特に各プロセスを周期的に起動するプロセススケジューリング方法に関する。
【０００２】
【従来の技術】従来、連続メディア処理を行なうプロセスのスケジューリング方法としては、Conductor/Performer モデル（1st International Workshop on Real-Tim

e Operating Systems and Applications,1994 ）を用いたスケジューリング方法が知られていた。連続メディア処理とは、画像や音声などをデジタル化したデータを変換、転送する処理を指す。

【０００３】この方法では、システムに存在するストリームが一つの場合、ストリームに属するプロセスが周期的にスケジューリングされることを保証する。ただし、ここで言うストリームとは、加工した連続メディアデータを直接または間接に受け渡しあうプロセス群を指す。同一ストリームに属するプロセスは、連続メディア処理データの加工処理を行なう順番が一意に決められている。先頭のプロセスは、外部装置（例えばビデオデジタイザ）から連続メディアデータの入力を得る。以後のプロセスは、順番が一つ前のプロセスから前のプロセスが加工した連続メディアデータを受け取り、自分が行なうべきデータ加工（例えば圧縮処理）を行ない、順番が一つ後のプロセスに自分が加工した連続メディアデータを送り渡す。最後尾のプロセスは、外部装置（例えばネットワークアダプタ）に連続メディアデータを出力する。

【０００４】Conductor/Performer モデルを用いたスケジューリング方法の概要を以下に示す。ストリームごとに、周期駆動の Conductor プロセスが一つ準備される。Conductor プロセスが起動すべき Performer プロセス（ストリームに属するプロセス）の順番は予め登録されている。Conductor プロセスは、この登録されている順番に従い Performer プロセスを起動する。そのために、Conductor プロセスおよび Performer プロセスは、共に自プロセスの起床通知用のメッセージキューを保持する。他プロセスの起動は、そのプロセスが保持するメッセージキューに対してメッセージを送信することにより行なう。

【０００５】Conductor プロセスは、指定された周期で駆動される。Conductor プロセスは、登録されている Performer プロセスの順番に従い、次の順番の Performer プロセスが所有するメッセージキューへのメッセージ送信と Conductor プロセスが所有者であるメッセージキューへのメッセージ到達待ちを行なうコール（関数呼び出し）を繰り返し発行する。すなわち、Conductor プロセスは、登録されている Performer プロセスの順番に従い、Performer プロセスの起動と、起動した Performer プロセスが一周期分の実行を完了するまでの休眠を繰り返す。最後の順番の Performer プロセスが実行を完了したら、Conductor プロセスは次の周期駆動のトリガがタイマ割り込みを契機にかけられるまで休眠する。

【０００６】一方、Performer プロセスは、Conductor プロセスからメッセージを受け取ることにより起床し、一周期分の連続メディア処理を行なう。一周期分の連続メディア処理を完了したら、Conductor プロセスが所有者であるメッセージキューにメッセージ送信と自プロセスが所有するメッセージキューへのメッセージ到達待ちを行なうコールを発行する。Performer プロセスは、次の周期の起動を通知するメッセージが Conductor プロセスから送信されるまで休眠することになる。

【０００７】上記スケジューリング方法は、システムに存在するストリームが一つの場合、Performer プロセスが周期的に CPU を割り当てられることを保証する。

【０００８】

【発明が解決しようとする課題】Conductor/Performer モデルに従ったスケジューリング方法は、システム上に存在するストリームが複数になった場合、以下の問題が生じる。

【０００９】（ａ）各 Conductor プロセス及び Performer プロセスの優先度が時間と共に変動しない。そのため、駆動周期が異なる Conductor プロセスが混在した場合、Conductor プロセスの実行開始間隔が変動する。すなわち Conductor プロセスが起床されたときに、同等の優先度又はより高い優先度をもつ他の Conductorプロセスや他の Performer プロセスが実行中である可能性がある。すなわち起床された Conductor プロセスと他の Conductor プロセス又は他の Performerプロセスとの間で CPU 時間の競合が生じ、これに伴い Conductor プロセスの実行開始間隔が変動するとともに後続する Performer プロセスの実行間隔も変動する。

【００１０】（ｂ）起床通知がプロセス間通信（IPC）により行なわれるため、メッセージ送信とメッセージ受信の関数呼び出しが発生し、起床通知に伴うオーバーヘッドが大きい。

【００１１】これらの問題点は、マルチメディアデータのリアルタイム MPEG 圧縮処理などの高スループットを要求される連続メディア処理の実現を困難にする。これらの処理では、連続メディアデータ入力時のバッファ管理を割り込みを使わずに行なわなければ、割り込みオーバーヘッドにより十分なスループットが得られなくなる。そのためには、システムに複数のストリームが存在する場合でも、Performer プロセスの実行間隔をできるだけ一定に保ち、割り込みによる通知なしに Performer プロセスが自発的に入力バッファの切り替えを行なわねばならない。同様に起床通知に用いる IPC のオーバーヘッドもスループットの低下を招く。

【００１２】また、Conductor/Performer モデルに従ったスケジューリング方法は、デッドラインミス（Conductor プロセスの駆動から指定された時間内で一周期分の処理を完了できなかった状態）を Conductor プロセスへのシグナル通知により行なう。シグナルハンドラの優先度は、通常、対象プロセスと同じ優先度を持つため、シグナルハンドラ処理により、他のストリームのプロセスの実行が遅延される可能性がある。すなわち一つのストリームの処理遅延が他ストリームの処理遅延を引き起こす可能性がある。

【００１３】本発明は、（ａ）複数ストリームがシステム内に存在する場合でも、連続メディア処理を行なうプロセスの実行間隔を一定に保ち、（ｂ）プロセスの起床、休眠の制御に伴うオーバーヘッドをより小さくし、（ｃ）処理遅延の回復処理を行なうシグナルハンドラを実行することによって、一つのストリームの処理遅延が生じても、他ストリームの処理遅延を引き起こさない、（ｄ）ネットワーク・パケットの到達など非同期イベントが発生した際に、連続メディア処理を行うプロセスの実行間隔の変動を防止する周期的プロセスのスケジューリング方法を提供する。

【００１４】

【課題を解決するための手段】本発明は、周期的に実行される少なくとも１つのプロセスを含むプロセスグループが複数存在し、これら複数のプロセスグループを並行実行させる周期的プロセスのスケジューリング方法において、各プロセスグループごとに起動間隔周期と一周期ごとの必要ＣＰＵ時間が指定されたとき、指定されたプロセスグループのＣＰＵ割り当て時間が他のプロセスグループのＣＰＵ割り当て時間と衝突しないように確保し、かつ指定された各プロセスグループの起動間隔周期を保つように調整する周期的プロセススケジューリング方法を特徴とする。

【００１５】また本発明は、プロセスグループの１つを起動する時点に達したとき、このプロセスグループに属するプロセスの実行優先度をシステム内で最高の優先度に変更することによって起動し、その後連続して割り当てられたＣＰＵ割り当て時間分だけ最高の優先度を保つプロセス起動方法を特徴とする。

【００１６】また本発明は、連続して割り当てられたＣＰＵ割り当て時間が経過しかつ必要ＣＰＵ時間を消費したとき、起動されたプロセスの実行優先度をシステム内で最高の優先度から基準の優先度に変更し、このプロセスにタイムアウトを通知するプロセス起動方法を特徴とする。

【００１７】さらに本発明は、システム内で最高の優先度をもつプロセスの実行中に、ネットワーク・パケットの到達など情報を受信すべき非同期イベントが発生したとき、直ちに最高優先度で走行中のプロセスの実行を停止し、情報の受信バッファを確保して情報を受信する準備を行った後に実行を停止したプロセスの実行を再開し、周期的に起動されるプロセスによって受信した情報を参照して処理する非同期イベント処理方法を特徴とする。

【００１８】

【発明の実施の形態】以下、本発明の実施形態について図面を用いて詳細に説明する。

【００１９】（１）第１の実施形態

本発明のスケジューリング方法を実現するためのプロセス起動の流れ、および連続メディアデータの流れを図１に示す。システムには一つの周期駆動カーネルプロセス(101)が存在する。周期駆動カーネルプロセス(101)は、タイマ割り込みハンドラ(104)により周期的に駆動される制御プロセスである。周期駆動カーネルプロセス(101)は、CPU 割り当て順序記述テーブル(900)を参照して次に連続メディアデータを処理するプロセス（以後周期プロセスと呼ぶ）(102)群を選択し、選択した周期プロセス(102)群の優先度を変更することにより、各周期プロセス(102)の周期的なスケジューリングを実現する。また、スケジューリングすべき周期プロセス(102)が存在しない場合には、それ以外の通常プロセス(109)をスケジューリングする。この動作の詳細は後述する。

【００２０】同一の連続メディアデータを処理する少なくとも１つの周期プロセス(102)は、一つのプロセスグループ(103)を形成する。プロセスグループ(103)に属する周期プロセス(102)は、その処理順が予め定められている。処理順が最初の周期プロセス(102)は、周期駆動カーネルプロセス(101)による優先度変更によって優先的に駆動され、外部入力装置(105)からの入力連続メディアデータを入力バッファ(106)を介して読み取り、データの加工を行なう。加工されたデータは共有バッファ(110)などを介して処理順が次の周期プロセス(102)に渡される。プロセスグループ(103)内の周期プロセス(102)の優先度は次々に継承され、処理順が最後の周期プロセス(102)は、出力バッファ(107)を介して外部出力装置(108)に出力し、自プロセスの優先度を下げることによって実質的にこのプロセスグループ(103)の１周期の処理を終了する。システムには複数のプロセスグループ(103)、例えば音声情報を処理するプロセスグループと画像情報を処理するプロセスグループなど、が存在し得る。

【００２１】なお図示しないスケジューラが関数呼び出しによって呼び出され、CPU 割り当て順序記述テーブル(900)を作成したり、指定されたプロセスを駆動する処理を行う。スケジューラは、呼び出されたプロセスによって動作しスケジューリングに関連する処理を行うプログラム・モジュールの集まりである。

【００２２】プロセスグループには、グループマスタプロセスが存在する。グループマスタプロセスは、プロセスグループに属するプロセスのうち、処理順が先頭であるプロセスである。プロセスグループに属するプロセスで、グループマスタプロセス以外のプロセスはスレーブプロセスと呼ばれる。プロセスグループ(103)の生成、削除は、グループマスタプロセスによって以下のインタフェースを用いて行われる。

【００２３】＜関数名＞

create_proc_group(master_pid, slave_pid_array,proc_array_number,pgroupid)

＜引数＞

master_pid: グループマスタプロセスのプロセス識別子

slave_pid_array: グループを構成するスレーブプロセ

ス識別子の配列
proc_array_number: グループを構成するスレーブプロセス識別子の数
pgroupid: 生成されたプロセスグループの識別子がリターンする。
<リターン値>
SUCCESS: 正常終了
もしくは、各種エラーコード
<説明>create_proc_group 関数は、master_pid で指定されるプロセスをグループマスタとするプロセスグループを生成する。生成されたプロセスグループは、master_pid で指定されるプロセスの他に、slave_pid_array, proc_array_number で指定されるプロセス群から構成される。pgroupid に生成されたプロセスグループ識別子が返る。なお master_pid 及び slave_pid_array で指定するプロセス識別子をもつ個々のプロセスはすでに生成済であることが前提である。

【００２４】<関数名>
destroy_proc_group(pgroupid)
<引数>
pgroupid: プロセスグループ識別子
<リターン値>
SUCCESS: 正常終了
もしくは、各種エラーコード
<説明>destroy_proc_group 関数は、pgroupid で指定されるプロセスグループを削除する。

【００２５】プロセスグループの管理に用いる配列データ及び制御ブロックのデータ構造を図２に示す。

【００２６】プロセスグループは、プロセスグループ制御ブロック(202)を用いて管理される。プロセスグループ制御ブロック(202)は、master_pid フィールド(203)、pid_array フィールド(204)、nproc フィールド(205)からなる。master_pid フィールド(203)は、プロセスグループのグループマスタプロセスのプロセス識別子を格納する。pid_array フィールド(204)は、プロセス識別子配列(206)へのポインタを格納する。プロセス識別子配列(206)は、プロセスグループを構成するスレーブプロセスのプロセス識別子の配列である。nproc フィールド(205)は、プロセス識別子配列(206)に格納されているプロセス識別子の数を格納する。また、プロセスグループ識別子からプロセスグループ制御ブロック(202)への変換は、プロセスグループ制御ブロックポインタ配列(201)を用いて行なう。すなわち、プロセスグループ制御ブロックポインタ配列(201)の、プロセスグループ識別子をインデックスに持つ要素に、プロセスグループ制御ブロック(202)へのポインタが格納されている。プロセスグループ識別子に対応するプロセスグループ制御ブロック(202)が存在しない場合には、プロセスグループ制御ブロックポインタ配列(201)の該当要素には、nil ポインタが格納されている。

【００２７】create_proc_group 関数の処理フローを図３に示す。

【００２８】ステップ301で、プロセス制御ブロックポインタ配列(201)の要素のうち、nilポインタが格納されている要素を一つ検索する。そのインデックス値を pgroupid のリターン値とする。

【００２９】ステップ302で、プロセスグループ制御ブロック(202)に用いるメモリ領域を確保する。

【００３０】ステップ303で、プロセス識別子配列(206)に用いるメモリ領域を確保する。

【００３１】ステップ304で、create_proc_group 関数の引数 master_pid で指定されたグループマスタプロセスの識別子を、ステップ302でメモリ領域を確保したプロセスグループ制御ブロック(202)の master_pid フィールド(203)に格納する。

【００３２】ステップ305で、create_proc_group 関数の引数 slave_pid_array で指定されたスレーブプロセスのプロセス識別子の配列を、ステップ303でメモリ領域を確保したプロセス識別子配列(206)にコピーする。

【００３３】ステップ306で、プロセス識別子配列(206)へのポインタをpid_array フィールド(204)に格納する。

【００３４】ステップ307で、create_proc_group 関数の引数 proc_array_number で指定されたプロセスグループを構成するスレーブプロセス識別子の数を、ステップ302でメモリ領域を確保したプロセスグループ制御ブロック(202)の nproc フィールド(203)に格納する。

【００３５】destroy_proc_group 関数の処理フローを図４に示す。

【００３６】ステップ401で、destroy_proc_group 関数の引数 pgroupid をインデックスに持つプロセスグループ制御ブロックポインタ配列(201)の要素を検索し、その要素により指されるプロセスグループ制御ブロック(202)が使用していたメモリ領域を解放する。

【００３７】ステップ402で、上記プロセスグループ制御ブロック(202)の pid_array フィールド(204)により指されるプロセス識別子配列(206)が使用していたメモリ領域を解放する。

【００３８】ステップ403で、プロセスグループ制御ブロックポインタ配列(201)の、解放するプロセスグループに対応する要素に nil ポインタを代入する。

【００３９】プロセスグループ(103)は、スケジューリングの単位になる。プロセスグループ(103)のグループマスタは、その初期化時において、alloc_time_slot 関数を用いて、指定周期ごとに指定時間にわたりプロセスグループ(103)に対し CPU が割り当てられることを予約する。また、CPU 時間の割り当てが不要になった場合には、dealloc_time_slot 関数を呼び出し、その予約を解除する。

【００４０】alloc_time_slot 関数が呼び出されると、

スケジューラは、各プロセスグループが要求する周期と1周期あたりの実行時間を満たすような CPU の割り当て順序を決定し、CPU 割り当て順序記述テーブル(900)を作成する。この作成アルゴリズムについては後述する。

【００４１】周期カーネルプロセス(101)は、CPU 割り当て順序記述テーブル(900)に基づいて各周期プロセス(102)のスケジューリングを行なう。プロセスグループ(103)に対して CPU を割り当てるべき時間が到達すると、周期駆動カーネルプロセス(101)はそのプロセスグループ(103)のグループマスタプロセスの優先度を raisedにする。優先度が raised の周期プロセス(102)は、ユーザプロセスの中で最高の優先度を持つことが保証される。また、優先度が raised のプロセス(102)は、周期駆動カーネルプロセス(101)よりも優先度が高いことも保証される。

【００４２】グループマスタプロセスの優先度が raised になってから指定時間経過すると、タイマ割り込みハンドラ(104)は、プロセスグループ(103)に属する周期プロセス(102)のうち優先度が raised である周期プロセス(102)の優先度を depressedにする（グループマスタプロセスは、同じプロセスグループ(103)に属する他の周期プロセス(102)に優先度を継承することが可能である。これについては後述する）。優先度が depressed のプロセス(102)は、ユーザプロセスの中で最低の優先度を持つことが保証される。

【００４３】これにより、プロセスグループ(103)に CPU が割り当てられるべき時間は、優先度が raised であるプロセスグループ(103)に属する周期プロセス(102)が実行可能状態にある限り、プロセスグループ(103)に属さないユーザプロセスや周期駆動カーネルプロセス(101)がスケジューリングされることはない。また、CPUが割り当てられるべきでない時間は、プロセスグループ(103)に属する周期プロセス(102)がスケジューリングされることはない。いずれの周期プロセス(102)にも割り当てられない CPU 時間は、通常プロセス(109)又は無限ループを実行し常に実行可能状態にあるアイドルプロセスに割り当てられる。アイドルプロセスの優先度を depressed に次いで低い優先度に設定することによって周期プロセス(102)又は周期駆動カーネルプロセス(101)に CPU が割り当てられるべきでない時間にスケジューリングされないことが保証される。

【００４４】alloc_time_slot 関数、dealloc_time_slot 関数の外部仕様は以下の通りである。

【００４５】<関数名>
alloc_time_slot(pgroupid, interval, length)
<引数>
pgroupid: CPU の割り当てを保証されるプロセスグループ識別子
interval: プロセスの起動間隔
length: 確保すべき一周期あたりのプロセスグループの実行時間
<リターン値>
SUCCESS: 正常終了
もしくは、各種エラーコード
<説明>alloc_time_slot 関数は、pgroupid で指定されるプロセスグループが、interval で指定される周期で、length で指定される時間にわたり CPU が割り当てられることを要求する。interval 及びlength は、所定のタイムスロットを単位として指定する。interval で指定される周期で、グループマスタプロセスの優先度が raised になる。グループマスタプロセスは、proc_raise_handoff 関数（後述）を用いて、プロセスグループに属する他のプロセスの優先度を raised にし、自プロセスを depressed （もしくは基準優先度）に変更することが可能である。グループマスタプロセスの優先度が raised になってから length で指定した時間が経過すると、プロセスグループに属するプロセスのうち優先度が raised になっているプロセスの優先度が強制的に depressed に変化する。さらにそのプロセスに対してタイムアウトシグナルが送信される。

【００４６】interval の値は2のべき乗でなければならない。それ以外の値が指定された場合には、指定値以下で最大の2のべき乗値が指定されたものとしてスケジューラは処理する。

【００４７】<関数名>
dealloc_time_slot(pgroupid)
<引数>
pgroupid: CPU の割り当て保証を解除するプロセスグループ識別子
<リターン値>
SUCCESS: 正常終了
もしくは、各種エラーコード
<説明>dealloc_time_slot 関数は、pgroupid で指定されるプロセスグループが保持していた CPU の割り当て要求を解除する。

【００４８】alloc_time_slot 関数により、CPU の割り当てを要求されたスケジューラは、すべてのプロセスグループ(103)の要求通りにプロセスグループ(103)のスケジューリングが行なえるとは限らない。図5に示す様に、同じ時刻に複数のプロセスグループ(501,502)をスケジューリングできないため、重なっているプロセスグループ(501,502)のうちいずれか一つのプロセスグループ(501)の CPU の割り当て時間(503)の重なっている時間を別の時間(504)にずらす必要が生じるためである。

【００４９】スケジューラは、以下のアルゴリズムに従い各プロセスグループ(103)の CPUの割り当て時間を決定し、その結果を後述の CPU 割り当て順序記述テーブル(900)に登録する。CPU の割り当てはタイマ割り込み発生間隔を単位に行なう。実時間を、タイマ割り込みの

発生する時刻を境界にタイムスロット群に分割する。以下のアルゴリズムに従い、各タイムスロットごとに割り当てるべきプロセスグループ(103)を決定していく。

【００５０】アルゴリズムのフローチャートを図６に示す。

【００５１】ステップ 601 において、割り当てるべきプロセスグループ(103)（alloc_time_slot 関数により、既に CPU の割り当て予約要求が発行されているプロセスグループ(103)）が要求している Interval 値のうち、最大の Interval 値と同じサイズを持つ図７に示すタイムスロットテーブル(700)を作成する。タイムスロットテーブル(700)は１次元の配列であり、配列の各要素には、対応するタイムスロットに割り当てられるべきプロセスグループ(103)識別子を格納していく。各要素の初期値として、該当タイムスロットが未割り当てであることを示す識別子が格納される。

【００５２】ステップ 602 において、割り当てるタイムスロットをまだ決定していないプロセスグループ(103)の存在を調べる。すべてのプロセスグループ(103)に対してタイムスロットの割り当てが完了している場合には正常終了（ステップ613）する。

【００５３】ステップ 603 において、まだ割り当てるタイムスロットを決定していないプロセスグループ(103)のうち、alloc_time_slot 関数発行時に要求した Interval 値が最小のプロセスグループ(103)を選択する。

【００５４】ステップ 604 において、ステップ 603 で選択されたプロセスグループ(103)の要求している Interval 値を Ｉ に、Length 値を Ｌ に代入する。

【００５５】ステップ 605 において、タイムスロット０～Ｉ－１までのうち、未割り当てのタイムスロットを、隣接しているタイムスロットごとにグループ化する。以後、本ステップでグループ化されたタイムスロット群 X_i (i=1,2,　n) を連続空きタイムスロットと呼ぶことにする。

【００５６】ステップ 606 において、ステップ 605 で得られた連続空きタイムスロット群のサイズ（タイムスロット数）の合計が Ｌ よりも少ない場合には、すべてのプロセスグループ(103)の要求を満たすタイムスロットの割り当てが不可能であると判断して異常終了（ステップ 614）する。

【００５７】ステップ 607 において、ステップ 605 で得られた連続空きタイムスロットのうち最大サイズをもつものと Ｌ の大小を比較する。

【００５８】Ｌ の方が小さければ、ステップ 608 において、Ｌ のサイズ以上のサイズを持つ連続空きタイムスロットのうち、サイズが最小のものを選択する。

【００５９】ステップ 609 において、ステップ 608 で選択した連続空きタイムスロットの先頭の Ｌ タイムスロットを、ステップ 603 で選択したプロセスグループ(103)に割り当てる。ここで割り当てられたタイムスロットの他に、I, 2I,3I... タイムスロット後のタイムスロットもステップ 603 に選択したプロセスグループ(103)に割り当てる。これで、ステップ 603 で選択したプロセスグループ(103)に対するタイムスロットの割り当てを完了し、ステップ 602 にジャンプする。

【００６０】ステップ 607 において、Ｌ の方が、ステップ 605 で得られた最大の連続空きタイムスロットサイズより大きければ、ステップ 610 において、サイズが最大の連続空きタイムスロットを選択する。

【００６１】ステップ 611 において、ステップ 610 で選択した連続空きタイムスロットに属する全タイムスロットをステップ 603 で選択したプロセスグループ(103)に割り当てる。さらに、ここで割り当てられたタイムスロットの他に、I,2I,3I ...タイムスロット後のタイムスロットもステップ 603 に選択したプロセスグループ(103)に割り当てる。

【００６２】ステップ 612 では、Ｌ からステップ 610 で選択した連続空きタイムスロットのサイズを引いた値を新しい Ｌ の値として、ステップ 605 にジャンプする。

【００６３】タイムスロットテーブル(700)の作成例を図７と図８を用いて示す。

【００６４】タイムスロットを割り当てるプロセスグループ(103)は３つとする。各プロセスグループ(103)が要求している Interval 値(801)、Length 値(802)を図８に示す。

【００６５】まず、３つのプロセスグループ(103)が要求している Interval 値(801)の最大値である 32 の大きさを持つタイムスロットテーブル(700)を作成する。タイムスロットテーブル(700)の各要素は、未割り当てを示す識別子に初期化する。

【００６６】Interval 値(801)が最小のプロセスグループＡに割り当てるタイムスロットを決定する。タイムスロット０～７のタイムスロット群から連続空きタイムスロットを生成する。この場合は、タイムスロット０～７からなるサイズ８の連続空きタイムスロットが一つ生成される。

【００６７】プロセスグループＡが要求している Length 値(802) 2は、先程生成した連続空きタイムスロットのサイズ８より小さいため、この連続空きタイムスロットの先頭２タイムスロット、すなわち、タイムスロット０と１がプロセスグループＡに割り当てられる。この他に、プロセスグループＡが要求している Interval 値(801) ８の整数倍後のタイムスロットもプロセスグループＡに割り当てられる。すなわち、タイムスロット０、１の他に、８、９、16、17、24、25がプロセスグループＡに割り当てられる。これに従い、タイムスロットテーブル(700)の該当要素を更新する。これでプロセスグループＡに対するタイムスロットの割り当ては完了する。

【００６８】次に、Interval 値(801)が２番目に小さい

プロセスグループBに割り当てるタイムスロットを決定する。タイムスロット0～15から、連続空きタイムスロットを生成する。この場合、タイムスロット2～7及びタイムスロット10～15からなる各サイズ6の連続空きタイムスロットが2つ生成される。

【0069】プロセスグループBが要求している Length 値(802)3は、このサイズ6より小さい。3以上のサイズを持つ連続空きタイムスロットのうちサイズが最小の連続空きタイムスロットを選択する。ここでは、タイムスロット2～7からなる連続タイムスロットが選択される。この連続空きタイムスロットの先頭3タイムスロットがプロセスグループBに割り当てられる。すなわち、タイムスロット2～4がプロセスグループBに割り当てられる。同様に、タイムスロット18～20もプロセスグループBに割り当てられる。これに従い、タイムスロットテーブル(700)の該当要素を更新する。これでプロセスグループBに対するタイムスロットの割り当ては完了する。

【0070】最後にプロセスグループCに割り当てるタイムスロットを決定する。タイムスロット0～31から、連続空きスロットを生成する。この場合、以下の連続空きタイムスロットが生成される。

・タイムスロット5～7からなるサイズ3の連続空きタイムスロット
・タイムスロット10～15からなるサイズ6の連続空きタイムスロット
・タイムスロット21～23からなるサイズ3の連続空きタイムスロット
・タイムスロット26～31からなるサイズ6の連続空きタイムスロット

プロセスグループCが要求している Length 値(802)7は、上記連続空きタイムスロットの最大サイズ6より大きい。そこで、まず、最大サイズ6を持つ連続空きタイムスロットを一つ選択する。ここではタイムスロット10～15からなる連続空きタイムスロットが選択されたとする。この連続タイムスロットに属するすべてのタイムスロット、すなわち、タイムスロット10～15がプロセスグループCに割り当てられる。これに従いタイムスロットテーブル(700)の該当要素を更新する。

【0071】プロセスグループCが要求している Length 値(802)7から、先に選択した連続空きタイムスロットのサイズ6を引いた残り1タイムスロットの割り当てを次に行なう。再び、連続空きスロットを生成する。この場合は、以下の連続空きタイムスロットが生成される。

・タイムスロット5～7からなるサイズ3の連続空きタイムスロット
・タイムスロット21～23からなるサイズ3の連続空きタイムスロット
・タイムスロット26～31からなるサイズ6の連続空きタイムスロット

1以上のサイズを持つ連続空きタイムスロットのうち、サイズが最小のものを選択する。この場合、タイムスロット5～7からなる連続空きタイムスロットが選択される。この連続空きタイムスロットの先頭1タイムスロット、すなわちタイムスロット5がプロセスグループCに割り当てられる。これに従い、タイムスロットテーブル(700)の該当要素を更新する。これでプロセスグループCに対するタイムスロットの割り当ては完了する。

【0072】以上の処理によってタイムスロットごとに割り当てられるプロセスグループは図7に示す通りに決定される。これから、スケジューラは図9に示す CPU 割り当て順序記述テーブル(900)を生成する。この CPU 割り当て順序記述テーブル(900)は、CPU を割り当てるべきプロセスグループ(915)の順序とその割り当て時間(916)（タイムスロット数）を記述したテーブルである。また、終了予定フラグ(917)は、その行の CPU 割り当てが完了したとき、プロセスグループ(915)の一周期分の割り当てが終了するか否かを示す。例えば、903 の終了予定フラグ(917)は、プロセスグループCにタイムスロット5を割り当てても一周期分の割り当てを完了していないため OFF あるいは FALSE になっているが、906 の終了予定フラグ(917)は、タイムスロット10～15を割り当てれば一周期分の割り当てを完了するため ON あるいは TRUE になっている。タイムスロットテーブル(700)からCPU 割り当て順序記述テーブル(900)への変換アルゴリズムは自明であるため省略する。インデックス(914)は、次に CPU を割り当てる CPU 割り当て順序記述テーブル(900)の行(エントリ)を示すポインタである。プロセスグループ(915)がOTHERS を指定するタイムスロットは、通常プロセス(109)に割り当てられるタイムスロットである。OTHERS(907)は、プロセスグループCに割り当てられた6タイムスロットが経過する前にプロセスグループC中の周期プロセスの実行が終了した場合に空いたタイムスロットが通常プロセス(109)に割り当てられることを示している。

【0073】スケジューラは、alloc_time_slot 関数が発行されるごとに図8のような既存のテーブルと新しい alloc_time_slot 関数による要求とを基にして CPU 割り当て順序記述テーブル(900)を作成し直す。

【0074】上記アルゴリズムに従って生成された CPU 割り当て順序記述テーブル(900)に従い、周期駆動カーネルプロセス(101)はプロセスグループ(103)のスケジューリングを行なう。周期駆動カーネルプロセス(101)が、周期的な CPU 割り当てを要求してきたプロセスグループ(103)に属する周期プロセス(102)の優先度を raised または depressed または基準優先度に変更することにより、このスケジューリングを実現する。

【0075】また、周期駆動カーネルプロセス(101)によって優先度が raised になった周期プロセスは、自プ

ロセスの優先度を depressed に変更し、かつ同一プロセスグループに属する他の周期プロセスの優先度を raised に変更することにより、プロセス・グループ内のハンドオフ・スケジューリングを実現する。

【００７６】優先度の変更は、proc_raise，proc_raise_cancel，proc_raise_handoff，proc_depress，proc_depress_cancel 関数を用いて行う。これら関数の外部仕様を以下に示す。

【００７７】＜関数名＞
proc_raise(pid, time, flags)
＜引数＞
pid: プロセス識別子
time: 優先度を raised に保つ時間
flags: 指定時間経過後の優先度を指定するフラグ。以下のフラグが指定可能である。
RIORITY_NORMAL
プロセスの優先度を基準優先度に変更する。
PRIORITY_DEPRESSED
プロセスの優先度を depressed に変更する。また、指定時間経過時に pid で指定されるプロセスにシグナルを送信するか否かを以下のフラグで指定する。
SEND_SIGNAL
指定時間経過時に pid で指定されるプロセスにタイムアウトシグナルを送信する。
＜リターン値＞
SUCCESS: 正常終了
もしくは、各種エラーコード
＜説明＞proc_raise 関数は、pid で指定されるプロセスの優先度を time で指定される時間（タイムスロット数）だけ raised に設定する。time にはそのプロセスが属するプロセスグループに割り当てられた時間(916)又は INFINITY を指定する。優先度が raised であるプロセスは、他のいかなるユーザプロセスよりも優先度が高いことが保証される。

【００７８】複数のプロセスの優先度を同時に raised にすることはできない。すでに優先度が raised であるプロセスが存在するときに本関数がコールされた場合、本関数はエラーリターンする。

【００７９】time に INFINITY が設定されている場合には、該当プロセスに対し proc_raise_cancel 関数か proc_raise_handoff 関数が発行されるまで、該当プロセスの優先度は raised に保たれる。INFINITY は、例えばタイマ割り込みハンドラ(104)が周期駆動カーネルプロセス(101)を起動するときに指定される。周期プロセス(102)を起動するときには、通常 INFINITY は指定されない。

【００８０】time に INFINITY 以外の値が指定されている場合には、time で指定した時間が経過しても、該当プロセスに対して proc_raise_cancel 関数か proc_raise_handoff 関数が発行されなければ、flags で指定されているフラグに応じて、プロセスの優先度が強制的に変更される。flags に PRIORITY_NORMAL が指定されていると、プロセスの優先度が raised から基準優先度に変更される。ｆｌａｇｓに　ＰＲＩＯＲＩＴＹ＿ＤＥＰＲＥＳＳＥＤ　が指定されていると、プロセスの優先度が raised からdepressed に変化する。さらに flags に SEND_SIGNAL が指定されていれば、そのプロセスに対してタイムアウトシグナルが送信される。タイムアウトシグナルを受信したプロセスは、設定された優先度に従って起動され、タイムアウトの場合の処理を行うことができる。

【００８１】＜関数名＞
proc_raise_cancel(pid, flags)
＜引数＞
pid: プロセス識別子
flags: 変更後の優先度を指定するフラグ。以下が指定可能である。
PRIORITY_NORMAL
プロセスの優先度を基準優先度に変更する。
PRIORITY_DEPRESSED
プロセスの優先度を depressed に変更する。
＜リターン値＞
SUCCESS: 正常終了
もしくは、各種エラーコード
＜説明＞proc_raise_cancel 関数は、proc_raise 関数により raised に設定されたプロセスの優先度を flags に応じて変更する。flags に PRIORITY_NORMAL が指定されている場合には、変更後の優先度は、スケジューリング属性に従って基準優先度になる。flags に PRIORITY_DEPRESSED が指定されている場合には、変更後の優先度は depressed になる。

【００８２】＜関数名＞
proc_raise_handoff(pid, flags)
＜引数＞
pid: プロセス識別子
flags: ハンドオフ後の優先度を指定するフラグ。以下が指定可能である。
PRIORITY_NORMAL
プロセスの優先度を基準優先度に変更する。
PRIORITY_DEPRESSED
プロセスの優先度を depressed に変更する。
＜リターン値＞
SUCCESS: 正常終了
もしくは、各種エラーコード
＜説明＞proc_raise_handoff 関数は、pid で指定されるプロセスの優先度をraised にし、かつ自プロセスの優先度を flags に応じて変更する。pid で指定されるプロセスは、呼び出しプロセスと同じプロセスグループに属していなければならない。そうでなければエラーリターンする。flags に PRIORITY_NORMAL が指定されて

いる場合には、ハンドオフ後の呼び出しプロセスの優先度は基準優先度になる。flags に PRIORITY_DEPRESSED が指定されている場合には、ハンドオフ後の呼び出しプロセスの優先度は depressed になる。

【００８３】呼び出しプロセスの優先度は raised でなければならない。優先度が raisedでないプロセスが本関数を呼び出すとエラーリターンする。

【００８４】呼び出しプロセスの優先度が raised に保たれる上限時間が指定されている場合（proc_raise 関数の time 引数に INFINITY 以外が指定されている場合）には、ハンドオフ先のプロセスの優先度が raised に保たれる時間は、呼び出し時点で残っている、ハンドオフ元のプロセスの優先度が raised に保たれる時間となる。呼び出しプロセスの優先度が raised に保たれる上限時間が指定されていない場合は、ハンドオフ先のプロセスの優先度が raised に保たれる上限時間も存在しない。

【００８５】<関数名>
proc_depress(pid, time, flags)
<引数>
pid: プロセス識別子
time: 優先度を depressed に保つ時間
flags: 指定時間経過後の優先度を指定するフラグ。以下のフラグが指定可能である。
PRIORITY_NORMAL
プロセスの優先度を基準優先度に変更する。
PRIORITY_RAISED
プロセスの優先度を raised に変更する。
<リターン値>
SUCCESS: 正常終了
もしくは、各種エラーコード
<説明>proc_depress 関数は、pid で指定されるプロセスの優先度を time で指定される時間（タイムスロット数）だけ depressed に設定する。優先度が depressed であるプロセスは、他のいかなるユーザプロセスよりも優先度が低いことが保証される。proc_depress 関数は、主として周期駆動カーネルプロセス(101)が自プロセスの優先度を下げて通常プロセス(109)を起動することを目的として発行される。

【００８６】time に INFINITY が設定されている場合には、該当プロセスに対し proc_depress_cancel 関数か proc_raise_handoff 関数が発行されるまで、該当プロセスの優先度は depressed に保たれる。

【００８７】time に INFINITY 以外の値が指定されている場合には、time で指定した時間が経過しても、該当プロセスに対して proc_depress_cancel 関数が発行されなければ、flags で指定されているフラグに応じて、プロセスの優先度が強制的に変更される。flags に PRIORITY_NORMAL が指定されていると、プロセスの優先度が depressed から基準優先度に変更される。flags に PRIORITY_RAISED が指定されていると、プロセスの優先度が depressed から raised に変化する。

【００８８】<関数名>
proc_depress_cancel(pid, flags)
<引数>
pid: プロセス識別子
flags: 変更後の優先度を指定するフラグ。以下が指定可能である。
PRIORITY_NORMAL
プロセスの優先度を基準優先度に変更する。
PRIORITY_RAISED
プロセスの優先度を raised に変更する。
<リターン値>
SUCCESS: 正常終了
もしくは、各種エラーコード
<説明>proc_depress_cancel 関数は、proc_depress 関数により depressed に設定されたプロセスの優先度を flags に応じて変更する。flags に PRIORITY_NORMAL が指定された場合には、変更後のプロセスの優先度は基準優先度になる。flagsに PRIORITY_RAISED が指定された場合には、変更後のプロセスの優先度は raised になる。

【００８９】上記関数群を実現するために必要な配列データ及び制御ブロックのデータ構造を図１０に示す。

【００９０】プロセス管理はプロセス制御ブロック(1002)を用いて行なわれる。実行可能（レディ）状態にあるプロセスのプロセス制御ブロック(1002)は、優先度別に、レディキューヘッダ配列(1001)の要素をキューヘッダに持つ双方向キュー（以後、双方向レディキューと呼ぶ）に接続される。レディキューヘッダ配列(1001)は、優先度をインデックス値とした、優先度別の双方向レディキューのキューヘッダ（プロセス制御ブロックを指すポインタ）の配列である。ただし、優先度は値が小さいほど高優先度であるものとする。また、最高優先度の値は raised、最低優先度の値は depressed で表す。

【００９１】プロセス制御ブロック(1002)は、優先度別の双方向レディキューに接続するために、next_proc フィールド(1003)および prev_proc フィールド(1004)を保持する。それぞれ、双方向キューの後ろのプロセス制御ブロック(1002)へのポインタ、前のプロセス制御ブロック(1002)へのポインタが格納される。但し、双方向レディキューの先頭プロセス制御ブロック(1002)の prev_proc フィールド(1004)には、レディキューヘッダ配列(1001)の要素へのポインタが格納される。また、双方向レディキューの末尾プロセス制御ブロック(1002)の next_proc フィールド(1003)には、nil ポインタが格納される。

【００９２】プロセス制御ブロック(1002)には、その他に、counter フィールド(1005)、flags フィールド(1006)、context フィールド(1007)、base_pri フィールド

(1010)が存在する。counter フィールド(1005)には、そのプロセスが raised もしくはdepressed の優先度を保てる残り時間（タイムスロット数）を保持する。flags フィールド(1006)には、raised もしくは depressed の優先度を保てる時間が経過した後に、変更されるべきプロセスの優先度を示すフラグが格納される。context フィールド(1007)は、プロセスの実行コンテクストの退避領域である。base_pri フィールド(1010)は、プロセスの基準優先度が、プロセス生成時に格納される。

【0093】プロセス識別子からプロセス制御ブロック(1002)への変換は、プロセス制御ブロックポインタ配列(1009)を用いて行なう。すなわち、プロセス制御ブロックポインタ配列(1009)の、プロセス識別子をインデックスに持つ要素に、プロセス制御ブロック(1002)へのポインタが格納されている。プロセス識別子に対応するプロセス制御ブロック(1002)が存在しない場合には、プロセス制御ブロックポインタ配列(1009)の該当要素には、nil ポインタが格納されている。

【0094】また、ctxproc(1008)は、現在実行中のプロセスのプロセス制御ブロック(1002)へのポインタを格納する。

【0095】proc_raise 関数の処理フローを図１１に示す。

【0096】ステップ1101で、スケジューラはプロセス制御ブロックポインタ配列(1009)の、proc_raise 関数の引数 pid をインデックスに持つ要素を求め、その要素から指されるプロセス制御ブロック(1002)を、双方向レディキューからデキューする。

【0097】ステップ1102で、レディキューヘッダ配列(1001)の raised をインデックスに持つ要素を求め、その要素をキューヘッダとする双方向レディキューの末尾に、ステップ1101で得られたプロセス制御ブロック(1002)をエンキューする。

【0098】ステップ1103で、proc_raise 関数の引数 time で指定した値と、ステップ1102で得られたプロセス制御ブロック(1002)の counter フィールド(1005)に格納する。

【0099】ステップ1104で、proc_raise 関数の引数 flags で指定した値を、ステップ1102で得られたプロセス制御ブロック(1002)の flags フィールド(1006)に格納する。

【0100】ステップ1105で、現在の実行コンテクスト（各種レジスタの値）を、ctxproc(1008)から指されるプロセス制御ブロック(1002)の context フィールド(1007)に退避する。

【0101】ステップ1106で、ctxproc(1008)に、システム内で最高優先度を持つプロセスのプロセス制御ブロック(1002)へのポインタを格納する。システム内で最高優先度を持つプロセスのプロセス制御ブロックは以下の手順で検索できる。まず、レディキューヘッダ配列(1001)に格納されている各双方向レディキューのうち、キュー長が１以上であり、かつ、インデックス値が最小の双方向レディキューを求める。その双方向レディキューの先頭にキューイングされているプロセス制御ブロック(1002)が、求めるプロセス制御ブロック(1002)になる。ここでは、proc_raise 関数の引数 pid で指定され、ステップ1102で raised の値をもってレディキューヘッダ配列(1001)に接続されたプロセスが最高優先度を持つプロセスである。

【0102】ステップ1107で、ステップ1106で得られたプロセス制御ブロックの contextフィールド(1007)に退避されている実行コンテクストを回復する。ステップ1107の処理によってプロセスのスイッチが生じ、実行コンテクストの回復されたプロセスがディスパッチされる。なおステップ1104とステップ1105との間でシステム内で最高優先度を持つプロセスのプロセス制御ブロックポインタと ctxproc(1008)の内容を比較し、両者が同じであればステップ1105～1107の処理をスキップできる。

【0103】proc_raise_cancel，proc_raise_handoff，proc_depress，proc_depress_cancel 関数も、ステップ1101から1102と同様のレディキュー操作、および、（必要ならば）ステップ1103から1104と同様のプロセス制御ブロック(1002)の各種フィールドの更新、および、ステップ1105と同様の実行コンテクストの退避、および、ステップ1106から1107と同様のシステム内で最高優先度を持つプロセスの実行コンテクストの回復、を行なうことで実現できる。処理フローは proc_raise 関数と同様になるため、省略する。

【0104】proc_raise 関数及びproc_depress 関数が発行された場合、その関数発行から、引数 time で指定された時間が経過したか否かの検査を行なう必要がある。この検査は、プロセス制御ブロックの counter フィールド(1005)を用いてタイマ割り込みハンドラ(104)内で行なわれる。またタイマ割り込みハンドラ(104)は、周期駆動カーネルプロセス(101)の駆動処理も併せて行う。これらを行うタイマ割り込みハンドラ(104)の処理フローを図１２に示す。あらかじめ設定したタイムスロットごとにタイマ割り込みが発生してタイマ割り込みハンドラ(104)に制御が渡り、図１２に示す処理が実行される。

【0105】ステップ1201では、レディキューヘッダ配列(1001)の raised をインデックスに持つ要素を求め、その要素に格納されているプロセス制御ブロック(1002)へのポインタを変数 PCB に代入する。

【0106】ステップ1202では、ステップ1201で更新された PCB の値が nil ポインタであるか否かのチェックを行なう。nil ポインタであればステップ1207に、nil ポインタでなければステップ 1203 にジャンプする。

【0107】ステップ1203では、PCB で指されるプロセス制御ブロック(1002)の counterフィールド(1005)を１

だけデクリメントする。ただし、counter フィールド(1005)に INFINITY が格納されている場合には、何も行なわない。

【0108】ステップ1204では、PCB で指されるプロセス制御ブロック(1002)の counterフィールド(1005)の値が0であるか否かのチェックを行なう。counter フィールド(1005)の値が0であればステップ1205に、0以外であればステップ1206にジャンプする。

【0109】ステップ1205では、PCB で指されるプロセス制御ブロック(1002)の flags フィールド(1006)に応じてレディキュー操作を行なう。すなわち以下の操作を行なう。まず、PCB で指されるプロセス制御ブロック(1002)を、双方向レディキューからデキューする。次に、flags フィールド(1006)に PRIORITY_NORMAL が格納されている場合には、レディキューヘッダ配列(1001)の base_pri フィールド(1010)に格納されている値をインデックスとする要素を求め、その要素をキューヘッダとする双方向レディキューの末尾に PCB で指されるプロセス制御ブロック(1002)をエンキューする。また、flags フィールド(1006)に PRIORITY_DEPRESSEDが格納されている場合には、レディキューヘッダ配列(1001)の depressed をインデックスとする要素を求め、その要素をキューヘッダとする双方向レディキューの末尾に PCB で指されるプロセス制御ブロック(1002)をエンキューする。

【0110】ステップ1206では、PCB の値を、PCB で指されているプロセス制御ブロック(1002)の next_proc フィールド(1003)の値に更新する。その後ステップ1202にジャンプする。

【0111】ステップ1207では、レディキューヘッダ配列(1001)の depressed をインデックスに持つ要素を求め、その要素に格納されている制御ブロック(1002)へのポインタを変数 PCB に代入する。

【0112】ステップ1208では、ステップ1207で更新された PCB の値が nil ポインタであるか否かのチェックを行なう。nil ポインタであればステップ1213に、nil ポインタでなければステップ 1209 にジャンプする。

【0113】ステップ1209では、PCB で指されるプロセス制御ブロック(1002)の counterフィールド(1005)を1だけデクリメントする。ただし、counter フィールド(1005)に INFINITY が格納されている場合には、何も行なわない。

【0114】ステップ1210では、PCB で指されるプロセス制御ブロック(1002)の counterフィールド(1005)の値が0であるか否かのチェックを行なう。counter フィールド(1005)の値が0であればステップ1211に、0以外であればステップ1212にジャンプする。

【0115】ステップ1211では、PCB で指されるプロセス制御ブロック(1002)の flags フィールド(1006)に応じてレディキュー操作を行なう。すなわち、以下の操作を行なう。まず、PCB で指されるプロセス制御ブロック(1002)を、双方向レディキューからデキューする。次に、flags フィールド(1006)に PRIORITY_NORMAL が格納されている場合には、レディキューヘッダ配列(1001)の base_pri フィールド(1010)に格納されている値をインデックスとする要素を求め、その要素をキューヘッダとする双方向レディキューの末尾に PCB で指されるプロセス制御ブロック(1002)をエンキューする。また、flags フィールド(1006)に PRIORITY_RAISED が格納されている場合には、レディキューヘッダ配列(1001)の raised をインデックスとする要素を求め、その要素をキューヘッダとする双方向レディキューの末尾に PCB から指されるプロセス制御ブロック(1002)をエンキューする。

【0116】ステップ1212では、PCB の値を、PCB で指されているプロセス制御ブロック(1002)の next_proc フィールド(1003)の値に更新する。その後、ステップ1208にジャンプする。

【0117】ステップ1213からステップ1216において、周期駆動カーネルプロセス(101)の駆動処理を行う。周期駆動カーネルプロセス(101)は、以下の事象が生じた場合に駆動される。

【0118】(a)プロセスグループ(915)に割り当てられるべき CPU 時間が経過したとき

周期駆動カーネルプロセス(101)は、CPU 割り当て順序記述テーブル(900)に従い、テーブルに記述された時間(916)を各プロセスグループ(915)に順次与えていく。プロセスグループ(915)に与えられた時間(916)が経過したとき、そのプロセスグループ(915)に属する優先度が raised のプロセスの PCB の counter が0になるので、ステップ1204及びステップ1205によってそのプロセスの優先度が PCB -> flags に応じてより低い優先度に変更され、結果として raised の次に優先度の高い周期駆動カーネルプロセス(101)が駆動される。周期駆動カーネルプロセス(101)は、CPU を割り当てるべきプロセスグループ(915)を変更する。次にCPU を割り当てるべきプロセスグループ(915)が存在しない場合、周期駆動カーネルプロセス(101)は自らの優先度を depressed に変更することにより、通常プロセス(109)への CPU 時間の割り当てを実現する。

【0119】(b)CPU の割り当てを要求しているプロセスグループ(915)の最小 Interval（例えば図8の例では8タイムスロットであり、以後最小 Interval と略す）が経過した場合

(a)で述べたように、周期駆動カーネルプロセス(101)は、次に CPU を割り当てるべきプロセスグループ(915)が存在しない場合、自プロセスの優先度を depressed に変更する。しかし、最小 Interval が経過するたびに、最小 Interval での駆動を要求しているプロセスグループ(915)に CPU 時間を割り当てる必要が生じる。そ

のため、最小 Interval 周期で周期駆動カーネルプロセス(101)を駆動し、該当プロセスグループ(915)に対するCPU 時間の割り当て処理を行う。

【0120】最小 Interval による周期駆動カーネルプロセス(101)の駆動間隔は、kproc_timer という変数によって管理される。この変数は、alloc_time_slot 関数によって CPU 割り当て順序記述テーブル(900)を作成するときスケジューラによって最小Interval に初期化される。

【0121】ステップ1213で、kproc_timer を１だけデクリメントする。

【0122】ステップ1214では、ステップ1213で更新された kproc_timer の値が０であるか否かをチェックする。０であれば、上記（ｂ）の周期駆動カーネルプロセス(101)の駆動契機であることを示す。周期駆動カーネルプロセス(101)の駆動処理のため、kproc_timer の再初期化後ステップ1215にジャンプする。

【0123】kproc_timer が０以外の場合には、ステップ1105からステップ1107で示される実行コンテクストの退避回復処理を実行する。プロセスグループ(915)に割り当てられるべき CPU 時間が経過した場合（上記（ａ）の場合）、周期駆動カーネルプロセス(101)がシステム内で最高優先度を保持するプロセスとなり、ステップ1105からステップ1107の実行によって周期駆動カーネルプロセス(101)が駆動される。

【0124】ステップ1215で、CPU 割り当て順序記述テーブル(900)の Index(914)を、プロセスグループ(915)のフィールドに OTHERS が格納されているエントリまで進める。例えば、図９で、Index(914)が906のエントリを指していた場合、本ステップはIndex(914)が907のエントリを指すように更新する。本ステップにより、ステップ1216で駆動される周期駆動カーネルプロセス(101)が、プロセスグループ(915)のフィールドに OTHERS が格納されているエントリの次のエントリから、プロセスグループ(915)への CPU 時間の割り当て処理を開始することができる。Index(914)がすべてにプロセスグループ(915)のフィールドに OTHERS が格納されているエントリを指している場合には、何も行わない。

【0125】ステップ1216で、proc_depress_cancel(kern_proc, PRIORITY_NORMAL)関数を呼び出す。kern_proc は、周期駆動カーネルプロセス(101)のプロセス識別子を表す。本関数の実行により、周期駆動カーネルプロセス(101)の優先度が depressed であれば depressed から基準優先度 (raised の次に高い優先度) に変更される。周期駆動カーネルプロセス(101)の優先度が基準優先度であれば、この関数の実行によって優先度の変更はない。この時点で、優先度が raised であるプロセスは存在しないことが保証される（各プロセスグループ(915)に、周期駆動カーネルプロセス(101)の駆動契機を待たないで CPU 時間を割り当てないように、CPU 割り当て順序記述テーブル(900)を設定している）ので、上記（ｂ）の場合の周期駆動カーネルプロセス(101)の駆動が実現できる。

【0126】なお周期プロセス(102)の PCB->counter がタイムスロットの間隔ごとに正しくデクリメントされるためには、周期プロセス(102)の PCB が常にレディキューに接続されている必要がある。従って周期プロセス(102)の PCB がレディキューから外されるようなウェイトは禁止される。周期プロセス(102)は、ウェイトする場合、割り当てられた CPU 時間内でダイナミック ジャンプ等の手段によってウェイトする必要がある。

【0127】周期駆動カーネルプロセス(101)の動作のフローチャートを図１３に示す。

【0128】周期駆動カーネルプロセス(101)は、CPU の割り当てを要求しているプロセスグループ(103)に割り当てられた時間(916)又は最小 Interval ごとに、タイマ割り込みハンドラ(104)によって駆動される。すなわちそのときタイマ割り込みハンドラ(104)を実行していたプロセスが周期駆動カーネルプロセス(101)を起動する。周期駆動カーネルプロセス(101)は、基準優先度で動作する。

【0129】ステップ 1301 において、CPU 割り当て順序記述テーブル(900)の Index(914)をテーブルの１エントリ分インクリメントする。CPU 割り当て順序記述テーブル(900)の Index(914) は、次に周期駆動カーネルプロセス(101)により CPUを割り当てられるべきプロセスグループ(915)などを示すエントリを指している。

【0130】ステップ 1302 において、Index(914) により指される CPU 割り当て順序記述テーブル(900)のエントリを検索する。

【0131】ステップ 1303 において、ステップ 1302 で得られたエントリのプロセスグループ(915)のフィールドが OTHERS であるか否かの検査を行なう。

【0132】プロセスグループ(915)が OTHERS である場合には、ステップ 1304において proc_depress(MYSELF, INFINITY, PRIORITY_NORMAL) をコールする。これにより、周期駆動カーネルプロセス(101)は proc_depress_cancel 関数が発行されるまで、優先度が depressed になる。周期的な CPU の割り当てを要求しているプロセスグループ(103)の最小 Interval ごとに、タイマ割り込みハンドラ(104)からproc_depress_cancel 関数が発行される。それまで、連続メディア処理を行なわない通常プロセス(109) (alloc_time_slot 関数により周期的なスケジューリングを要求していないプロセス) がスケジューリングされる。

【0133】プロセスグループ(915)が OTHERS でない場合には、ステップ 1305 において、次に CPU を割り当てるべきプロセスグループ(915)が一周期分の実行を完了しているか否かを検査する。これは、図１４に示すような、実行状態記述テーブル(1400)の done フィール

ド(1401)のフラグを用いて判定される。これは、alloc_time_slot 関数を用いて周期的なスケジューリングを要求しているプロセスグループ(103)ごとに、一周期分の実行が完了したか否かを記述するフィールドである。周期的なスケジューリングを要求しているプロセスグループ(103)に属している周期プロセス(102)は、一周期分の実行完了時に proc_raise_cancel 関数を自プロセスに対して発行する（後述）。この関数を呼び出した周期プロセス(102)が属するプロセスグループ(103)の done フラグは、この関数の処理ルーチン内でセットされる。

【０１３４】ステップ 1302 で得られたエントリのプロセスグループ(915)が一周期分の実行を完了していた場合には、ステップ 1306 においてそのエントリの終了予定フラグ(917)を検索する。

【０１３５】終了予定フラグ(917)が FALSE であれば、ステップ 1301 に戻る。

【０１３６】終了予定フラグ(917)が TRUE であれば、ステップ 1308 において実行状態記述テーブル(1400)の done フィールド(1401)の該当フラグをクリアする。さらにactive pid フィールド(1402)も初期化する。この初期化方法はすぐ後で述べる。この後、ステップ 1301 に戻る。

【０１３７】ステップ 1305 で一周期分の実行を完了していないと判定された場合には、ステップ 1309 において、ステップ 1302 で得られたエントリの終了予定フラグ(917)を検査する。

【０１３８】終了予定フラグ(917)が TRUE であれば、ステップ 1307 で proc_raise(pid,TIME, PRIORITY_DEPRESS | SEND_SIGNAL) を発行する。pid には、実行状態記述テーブル(1400)の active pid フィールド(1402)に格納されているプロセス識別子が使用される。TIME にはエントリの時間フィールド(916)の値が使用される。実行状態記述テーブル(1400)の active pid フィールド(1402)は、現在、プロセスグループ(103)に属するどの周期プロセス(102)が連続メディア処理を実行中であるかを示す。このステップの直後に、TIME で指定した時間にわたり pid で指定した周期プロセス(102)がスケジューリングされる。

【０１３９】ステップ 1308 で該当エントリが初期化される。ステップ 1308 では、ステップ 1302 で選択されたプロセスグループ(915)に対応する active pid フィールド(1402)内のエントリが、そのプロセスグループ(915)のグループマスタのプロセス識別子に初期化される。また、proc_raise_handoff 関数が発行されると、その処理ルーチン内で、ハンドオフ先のプロセス識別子に active pid フィールド(1402)の該当エントリが更新される。また proc_raise_cancel 関数が発行されると、終了予定フラグ(917)が TRUE の場合、その処理ルーチン内で active pid フィールド(1402)の該当エントリが初期化される。

【０１４０】ステップ 1309 で終了予定フラグ(917)が FALSE であると判定されれば、ステップ 1310 において、proc_raise(pid, TIME, PRIORITY_DEPRESS) を発行し、処理を終了する。pid, TIME の設定方法はステップ 1307 の場合と同様である。やはり、このステップの直後に、TIME で指定した時間にわたり pid で指定した周期プロセス(102)がスケジューリングされる。

【０１４１】周期的なスケジューリングを要求しているプロセスグループ(103)に属する周期プロセス(102)群の動作を図１５から図１７に示す。

【０１４２】図１５は、プロセスグループ(103)内に属する周期プロセス(102)の起動順序を示す図である。

【０１４３】プロセスグループ(103)に属する周期プロセス(102)は、その処理順が予め定められている。プロセスグループ(103)のグループマスタプロセス(1501)は、周期駆動カーネルプロセス(101)の proc_raise 関数により優先度が raised になり、起動する。プロセスグループ(103)に属する各プロセス(102)は、proc_raise_handoff 関数を用いて、次の順番の周期プロセス(102)に優先度を継承する。継承後の自プロセス(102)の優先度は depressed になる。最後の順番の周期プロセス(102)は、proc_raise_cancel 関数を用いて、一周期分の実行を完了する。

【０１４４】図１６は、グループマスタプロセス(1501)の動作を示すプログラムである。

【０１４５】1601 行目で、自プロセスをグループマスタプロセス(1501)とするプロセスグループ(103)を生成する。以後、このプロセスグループ(103)がスケジューリングの単位になる。

【０１４６】1602 行目で、1601 行目で生成したプロセスグループ(103)に、interval で指定される間隔で、length で指定される CPU 時間を割り当てることを要求する。この関数発行後、図１５で示されるように、周期駆動カーネルプロセス(101)からグループマスタプロセス(1501)に対し proc_raise 関数が発行されるようになる。

【０１４７】1603 行目から 1606 行目までが、連続メディア処理を行なうループである。一周期分の連続メディア処理を行なった後、1605 行目で proc_raise_handoff関数を呼び出し、次に処理を行なうべき 1601 行目で生成したプロセスグループ(103)に属するプロセス(102)の優先度を raised にする。自プロセスの優先度はdepressed になる。

【０１４８】連続メディア処理の実行ループを指定回数実行すると、1607 行目で、1602 行目で発行した CPU 時間の割り当て要求を解除する。

【０１４９】さらに、1608 行目で、1601 行目で生成したプロセスグループ(103)を削除する。

【０１５０】図１７は、スレーブプロセス(102)の動作を示すプログラムである。

【０１５１】1702 行目から 1704 行目までが連続メディア処理を行なうループである。

【０１５２】一周期分の連続メディア処理を行なった後、1703 行目で proc_raise_handoff関数を呼び出し、次に処理を行なうべき 1601 行目で生成したプロセスグループ(103)に属する周期プロセス(102)の優先度を raised にする。自プロセスの優先度は depressed になる。ただし、最後の順番の周期プロセス(102)は、1704 行目で proc_raise_cancel 関数を発行し、自プロセスの優先度を depressed に変更する。連続メディア処理の実行ループを指定回数実行すると、プログラムは終了する。

【０１５３】一周期分の CPU の割り当て時間内に一周期分の実行が完了しなかった場合には、実行状態記述テーブル(1400)の active pid フィールド(1402)に登録されているプロセス(102)に対しタイムアウトのシグナルが送信される。かつ、実行状態記述テーブル(1400)の done フィールド(1401)内フラグに、IN_SIGNAL_TRANSACTION を示すフラグが立てられる。

【０１５４】このフラグが立っているプロセスグループ(103)に関しては、ステップ 1305におけるプロセスグループ(915)の実行終了判定が常に TRUE と判定される。また、ステップ 1308 の done フィールド(1401)のクリアも行なわれない。すなわち、シグナルハンドラは、周期プロセス(102)の基準優先度で、通常プロセス(109)と同様にスケジューリングされ実行されることになる。これにより、一つのストリームの処理遅延が他ストリーム処理に影響を及ぼさないことを保証できる。

【０１５５】第１の実施形態によれば、単一の周期駆動カーネルプロセス(101)が CPU 割り当て順序記述テーブル(900)に基づいてすべてのプロセスグループ(103)の周期的スケジューリングをするので、複数のプロセスグループ(103)の間で CPU 時間の競合が生じることによって周期プロセス(102)の実行が遅延することはない。また周期プロセス(102)の実行優先度を変更した後、プロセス ディスパッチに依存した周期プロセス(102)の起動を行うので、プロセス間通信を介して周期プロセスを起動する場合に比べて起床通知に伴うオーバヘッドがより小さい。また指定した CPU 時間を使い果たし、タイムアウトとなったプロセスのシグナルハンドラ処理は、そのプロセスの基準優先度で実行されるため、シグナルハンドラ処理によって他のプロセスグループの実行遅延を引き起こすことはない。

【０１５６】（２）第２の実施形態

第１の実施形態では、プロセスグループ(103)に割り当てられた時間(916)又は最小 Interval ごとにタイマ割り込みハンドラ(104)を実行するプロセスから周期駆動カーネルプロセス(101)へプロセス スイッチするので、このときプロセス スイッチのオーバヘッドが介入する。第２の実施形態は、周期駆動カーネルプロセス(101)のかわりにプロセスのスケジューリングを制御するモジュール（以後、スケジューラと呼ぶ）を設けてタイマ割り込みハンドラ(104)とスケジューラを同一プロセスで実行することによってタイマ割り込みハンドラ(104)から周期駆動カーネルプロセス(101)へのプロセス スイッチのオーバヘッドを削減する。スケジューラを用いて本発明を実現するシステムの構成を図１８に示す。

【０１５７】システムには、一つのスケジューラ(1801)が存在する。スケジューラは、周期プロセスの優先度変更、次にスケジューリングするプロセスの決定、及びそのプロセスのディスパッチ動作を行なう。スケジューラは、タイマ割り込みハンドラ(104)により周期的に呼び出される。また、連続メディア処理を行なう周期プロセス(102)が、処理順が次の周期プロセス(102)に優先度の継承を行なう際、また、処理順が最後の周期プロセス(102)が、１周期分の実行を完了し、自プロセスの優先度を変更する際にも、各関数の処理ルーチン内でスケジューラは呼び出される。これらのスケジューラ呼び出し方法の詳細は後述する。

【０１５８】第１の実施形態と同様に、同一の連続メディア処理データを処理する周期プロセス(102)は、プロセスグループ(103)を形成する。プロセスグループ(103)の生成、削除は、前述の create_proc _group、destroy _proc_group 関数を用いて行なう。プロセスグループ(103)内で処理順の先頭の周期プロセス(102)は、外部入力装置(105)から入力バッファ(106)を介して連続メディアデータを読み取り、データ加工を行なう。加工されたデータは共有バッファ(110)を介して、処理順が次の周期プロセス(102)に渡される。処理順が最後の周期プロセス(102)は、出力バッファ(107)を介して外部出力装置(108)に出力する。

【０１５９】プロセスグループ(103)は、スケジューリングの単位となる。プロセスグループ(103)のグループマスタプロセスは、その初期化時において、前述の alloc_time_slot 関数を用いて、指定周期ごとに指定時間にわたりプロセスグループ(103)に対し CPU が割り当てられることを予約する。また、CPU 時間の割り当てが不要になった場合には、前述の dealloc_time_slot 関数を呼び出し、その予約を解除する。

【０１６０】alloc_time_slot 関数が呼び出されると、スケジューラは、各プロセスグループが要求する周期と１周期あたりの実行時間を満たすような CPU の割り当て順序を決定し、CPU 割り当て順序記述テーブル(900)を作成する。この作成アルゴリズムは第１の実施形態で述べた通りである。

【０１６１】スケジューラ(1801)は、CPU 割り当て順序記述テーブル(900)に基づいて各周期プロセス(102)のスケジューリングを行なう。プロセスグループ(103)に対して CPU を割り当てるべき時間が到達すると、スケジューラ(1801)はそのプロセスグループ(103)のグループマ

スタプロセスの優先度を raised にする（前述の proc_raise 関数を呼び出す），グループマスタプロセスの優先度が raised になってから指定時間経過すると、タイマ割り込みハンドラ(104)から呼び出されたスケジューラ(1801)は、プロセスグループ(103)に属する周期プロセス(102)のうち優先度が raised である周期プロセス(102)の優先度を depressed にする。このタイマ割り込みハンドラ(104)からのスケジューラの呼び出し方法、スケジューラの動作の詳細は後述する。これにより、プロセスグループ(103)に CPU が割り当てられるべき時間は、優先度が raised であるプロセスグループ(103)に属する周期プロセス(102)が実行可能状態にある限り、プロセスグループ(103)に属さないユーザプロセスがスケジューリングされることはない。また、CPU が割り当てられるべきでない時間は、プロセスグループ(103)に属する周期プロセス(102)がスケジューリングされることはない。いずれのプロセスグループ(103)にも割り当てられない CPU 時間は、通常プロセス(109)又はアイドルプロセスに割り当てられる。

【０１６２】上記で述べたように、スケジューラ(1801)は、タイマ割り込みハンドラ(104)または、自プロセスもしくは他プロセスの優先度変更を要求する周期プロセス(102)から呼ばれ得る。スケジューラを呼び出すときに用いるコマンドリストの形式を図１９に示す。

【０１６３】スケジューラ(1801)を呼び出すタイマ割り込みハンドラ(104)、周期プロセス(102)は、その呼び出し関数の引数として図１９で示されるコマンドリストの先頭エントリへのポインタ(1901)を指定する。コマンドリストは、スケジューラの動作の指示をリスト形式で示したものである。コマンドリストの各エントリは、next_command フィールド(1902)、flag フィールド(1903)、pid フィールド(1904)からなる。next_command フィールド(1902)は、次のエントリへのポインタが格納される。リストの最後尾のエントリの next_command フィールドの値は nilである。flag フィールドには、HANDOFF、CANCEL、INTERVAL、TIMER が指定可能である。また、pid フィールドは、flag フィールドが HANDOFF のときのみ意味を持つ。

【０１６４】タイマ割り込みハンドラ(104)は、その駆動のたびに flag に TIMER を格納したエントリからなるコマンドリストをスケジューラ(1801)に渡す。また、CPU の割り当てを要求しているプロセスグループ(103)の最小 Interval ごとに、上記エントリの他に、flag に INTERVAL を格納したエントリをもスケジューラに渡す。従ってタイマ割り込みハンドラ(104)は前述の kproc_timer によって最小 Interval を判定する必要がある。

【０１６５】周期プロセス(102)が、処理順が次の周期プロセス(102)に優先度を継承する場合、flags に HANDOFF を、pid に処理順が次の周期プロセス(102)の識別子を格納したエントリからなるコマンドリストをスケジューラに渡す。

【０１６６】また、処理順が最後の周期プロセス(102)が一周期分の実行を完了し、自プロセスの優先度を depressed に変更する場合は、flags に CANCEL を格納したエントリからなるコマンドリストをスケジューラに渡す。

【０１６７】最後に、上記コマンドリストを受け取り駆動するスケジューラの動作フローを図２０を用いて説明する。

【０１６８】ステップ2001において、スケジューラは、渡されたコマンドリストの先頭エントリの flag フィールドを検索する。flag フィールドが HANDOFF であればステップ2002に、flag フィールドが CANCEL であればステップ2003に、ジャンプする。flag フィールドが INTERVAL であればステップ1215を実行後、ステップ2006にジャンプする。flag フィールドが TIMER であればステップ1201～1212を実行した後、ステップ1204で PCB -> counter が０と判定されたプロセス、すなわちプロセスグループ(915)に割り当てられた CPU の時間(916)が経過したプロセスであり、かつ next_command フィールドによって指される flag フィールドが INTERVAL でなければステップ2015にジャンプする。この条件を満足しない場合にはステップ2012にジャンプする。

【０１６９】ステップ2015において、スケジューラ起動直前に CPU 時間が割り当てられていたプロセスグループ（ CPU 割り当て順序記述テーブル(900)の Index(914)で指されるエントリ）の終了予定フラグ(917)を検索する。終了予定フラグ(917)がON であれば、第１の実施形態で述べたタイムアウトのシグナル送信処理を行い、ステップ2006にジャンプする。終了予定フラグ(917)が OFF であれば直接ステップ2006にジャンプする。

【０１７０】ステップ2002において、優先度の継承元となる周期プロセス(1202)及び継承先となる周期プロセス(1202)の優先度、プロセス制御ブロック(1002)内のカウンタフィールド(1005)、フラグフィールド(1006)を更新する。この更新方法は、図１１で示したフローチャートと同様になるので省略する。そして、ステップ2012にジャンプする。

【０１７１】ステップ2003において、スケジューラ起動直前に CPU 時間が割り当てられていたプロセスグループ（CPU 割り当て順序記述テーブル(900)の Index(914)で指されるエントリ）の終了予定フラグ(917)を検索する。終了予定フラグ(917)が ON であればステップ2005に、OFF であればステップ2004にジャンプする。

【０１７２】ステップ2004において、実行状態記述テーブル(1400)の該当エントリの doneフィールド(1401)をセットしてステップ2006にジャンプする。

【０１７３】ステップ2005は、ステップ1308と同じ動作を行ない、ステップ2006にジャンプする。

【０１７４】ステップ2006は、ステップ1301～1302と同じ動作を行ない、ステップ2007にジャンプする。

【０１７５】ステップ2007において、ステップ2006で得られたエントリのプロセスグループ(915)のフィールドがOTHERSであるか否かの検査を行なう。OTHERSであったらステップ2012に、それ以外であったらステップ2008にジャンプする。

【０１７６】ステップ2008において、ステップ2006でインクリメントされたIndex(914)で指されるエントリの終了予定フラグ(917)、及び、そのエントリに対応する実行状態記述テーブル(1400)のdoneフィールド(1401)を検索する。両ビットが共にセットされていたらステップ2009に、doneフィールドのみセットされていたらステップ2006に、終了予定フラグのみセットされていればステップ2010に、共にクリアされていたらステップ2011にジャンプする。

【０１７７】ステップ2009は、ステップ1308と同じ動作を行ない、ステップ2006にジャンプする。

【０１７８】ステップ2010は、Index(914)で指されるエントリに登録されているプロセスグループ(103)のグループマスタプロセス(1501)である周期プロセス(102)を、エントリの時間(916)のフィールドで指定された時間にわたり優先度をraisedに変更し、かつ、その時間の経過後にシグナルを送信すべく、プロセスの優先度、及び対応するプロセス制御ブロック(1002)のカウンタフィールド(1005)、フラグフィールド(1006)の更新を行なう。この更新方法は、図１１で示したフローチャートと同様になるので省略する。その後ステップ2012にジャンプする。

【０１７９】ステップ2011は、Index(914)で指されるエントリに登録されているプロセスグループ(103)のグループマスタ(1501)である周期プロセス(102)を、エントリの時間フィールド(916)で指定された時間にわたり優先度をraisedに変更し、かつ、その時間の経過後にシグナルを送信しないように、プロセスの優先度、対応するプロセス制御ブロック(1002)のカウンタフィールド(1005)、フラグフィールド(1006)の更新を行なう。この更新方法は、図１１で示したフローチャートと同様になるので省略する。その後ステップ2012にジャンプする。

【０１８０】ステップ2012では、ステップ2001から2011で処理したエントリのnext_commandフィールドを検索する。その値がnilでなければステップ2001に戻る。nilであれば、ステップ1105から1107を実行後終了する。

【０１８１】（３）第３の実施形態

第１の実施形態又は第２の実施形態の周期プロセスのスケジューリング方法に従ってシステムが動作中にネットワーク・パケットの到達などの非同期イベントが多発すると、周期プロセス(102)の実行が阻害され、周期プロセス(102)の駆動周期間隔のゆらぎが大きくなる。すなわちネットワーク・パケットの受信処理などの非同期イベント処理は、ある程度の応答性能が求められるため、非同期イベント処理を優先し、その処理を行なうプロセスの優先度を周期プロセス(102)より高くすると、周期プロセス(102)の実行時間に遅れが生じ得る。第３の実施形態は、この問題を解決するために、ネットワーク・パケットの受信処理を行う割り込みハンドラを階層化するものである。第１の実施形態又は第２の実施形態のタイマ割り込みハンドラ(104)、周期駆動カーネルプロセス(101)、スケジューラ(1801)及びプロセスグループ(103)には変更がなく、第３の実施形態は第１の実施形態又は第２の実施形態を補強する形で実施される。以下非同期割り込みハンドラとしてネットワーク・パケットの受信処理を行なう割り込みハンドラを例にとってこの割り込みハンドラの構成、及びその動作方法を説明する。

【０１８２】図２１に、本割り込みハンドラを中心とするシステムの構成、及び割り込みハンドラの構成を示す。タイマ割り込みハンドラ(104)、周期駆動カーネルプロセス(101)、スケジューラ(1801)等は図示を省略する。

【０１８３】本システムは、ハードウェアとしてCPU(2101)と、Etherボード(2102)を有する。Etherボード(2102)は、パケットを受信した際に、そのパケット到達をCPU(2101)に通知し、パケット受信処理を行なうルーチン（割り込みハンドラ）をCPU(2101)上で駆動する機能を備える。また、CPU(2101)上には、パケット受信によって駆動される第１レベル割り込みハンドラ(2103)の他に、第２レベル割り込みハンドラ(2104)、アプリケーション・プログラム(2105)が設けられる。第２レベル割り込みハンドラ(2104)は、周期プロセス(102)の一つとして、前述したスケジューリング方法に従って周期駆動カーネルプロセス(101)又はスケジューラ(1801)により周期的に駆動される。アプリケーション・プログラム(2105)は、受信したパケットを処理する業務プログラムであり、周期プロセス(102)又は通常プロセス(109)によって走行する。

【０１８４】第１レベル割り込みハンドラ(2103)の動作フローを図２２に示す。

【０１８５】先に述べたように、第１レベル割り込みハンドラ(2103)は、Etherボード(2102)のパケット到達通知を契機に駆動される。そしてまずステップ2201で、受信したパケットをパケットキュー(2106)にエンキューする。ステップ2202で、空きバッファ群(2108)の中から受信バッファを一つ確保する。ステップ2203で、ステップ2202で確保した受信バッファに対するパケット受信を要求するコマンドをEtherボード(2102)に対し発行する。すなわち第１レベル割り込みハンドラ(2103)は、受信バッファを確保してパケットを受信する準備を行うだけであり、受信バッファ内の情報を参照するような処理を一切しない。Etherボード(2102)は、パケット到達を通知してから、ステップ2203が実行されるまでの間に到達し

たパケットを受信することができない。なぜなら、第1割り込みハンドラ(2103)から、受信パケットを格納すべき受信バッファのアドレスを指定されていないためである。従ってその間にパケットが到達しても、Etherボード(2102)はパケット受信に失敗し、そのパケットは喪失されるが、その時間は最小限に抑えられる。

【0186】第2レベル割り込みハンドラ(2104)は、指定した周期で周期駆動される。そして、第1レベル割り込みハンドラ(2103)によりエンキューされたパケットキュー(2106)に繋がれたパケットをデキューし、デキューしたパケットを参照してプロトコル処理を行なう。そして、このプロトコル処理の結果、アプリケーション・プログラム(2105)に渡すべき受信データが得られたならば、受信データキュー(2107)にその受信データをエンキューする。従って第2レベル割り込みハンドラ(2104)は、単にあるプロセスグループ(103)に属する周期プロセス(102)の一つとして、スケジューリングされる。

【0187】アプリケーション・プログラム(2105)は、受信データキュー(2107)にエンキューされた受信データのデキューと、受信データが格納されている受信バッファの解放処理を行なう。

【0188】周期プロセス(102)実行中にパケットが到達した場合、周期プロセス(102)の実行は停止され、第1割り込みハンドラ(2103)の実行が開始される。第1割り込みハンドラ(2103)の実行を最優先で行なうことにより、Etherボード(2102)のパケット受信失敗によるパケット喪失の発生を防いでいる。すなわち第1の実施形態又は第2の実施形態に第3の実施形態を適用することによって、従来のように第1レベル割り込みハンドラ(2103)と第2レベル割り込みハンドラ(2104)を1つのパケット受信割り込みハンドラで処理する場合に比べて、パケット喪失の確率を小さくするとともに、周期プロセス(102)の実行遅れを少なくし、この両者の利点を両立させるものである。

【0189】第2レベル割り込みハンドラ(2104)は、周期プロセス(102)としてスケジューリングされるため、自プロセスにCPU(2101)が割り当てられる時刻になるまでは、パケットキュー(2106)にパケットがキューイングされていても、その実行を開始することはない。そのため、第1割り込みハンドラ(2103)の実行が完了したら直ちに周期プロセス(102)の実行は再開される。このように割り込みハンドラを階層化することにより、パケット到達に伴う周期プロセス(102)の実行停止時間を、第1レベル割り込みハンドラ(2103)の実行時間のみに抑えることが可能になっている。

【0190】また、第2レベル割り込みハンドラ(2104)は周期駆動が保証されている。そのため、パケットが到達してからその駆動周期に相当する時間が経過する間に、必ず1回は第2レベル割り込みハンドラ(2104)が駆動される。すなわち、パケット到達から受信データキュー(2107)へのエンキューまでに要する時間の上限も第2レベル割り込みハンドラ(2104)の駆動周期となり、ネットワーク・パケットの受信処理の応答性能の保証も可能となる。

【0191】

【発明の効果】本発明のスケジューリング方法は、複数の周期的な CPU の割り当て要求を同時に満たす CPU 時間の割り当てアルゴリズムを提供する。このアルゴリズムを用いた、一周期分の連続メディア処理の実行開始間隔の変動は、周期駆動カーネルプロセスの駆動間隔より短くなることが保証される。かつ、要求周期の短いプロセス、すなわち実行開始間隔の変動の絶対値を小さく抑えねばならないプロセスほど周期駆動カーネルプロセスの駆動後すぐに CPU 時間が割り当てられる。そのため、周期の短いプロセスほど、実行開始間隔の変動を小さく抑えることが可能になる。

【0192】連続メディアデータの到達レートが一定である場合、入力バッファの切り替えなどの入力バッファ管理は周期的に行なえば良い。そのため、周期的なプロセスのスケジューリングが保証できれば、連続メディア処理を行なうプロセスが自発的に入力バッファを切り替えれば良く、入力連続メディアデータ到達の割り込みによる通知を必要としなくなる。割り込みオーバーヘッド削減による連続メディア処理の性能向上が期待できる。

【0193】また、本発明のスケジューリング方法では、連続メディア処理を行なう各プロセスの起床、休眠は優先度の変更により実現している。従来の IPC を用いた方式よりも、起床、休眠に要するオーバーヘッドが削減できる。この点からも連続メディア処理の性能向上が期待できる。

【0194】さらに、本発明のスケジューリング方法では、一つのプロセスグループにデッドラインミスが発生し、対象プロセスにシグナルが配送された場合、シグナルハンドラの優先度は、連続メディア処理を行なうプロセスよりも低優先度であることが保証される。そのため、一つのストリームの処理遅延が他ストリーム処理に影響を及ぼさないことを保証できる。

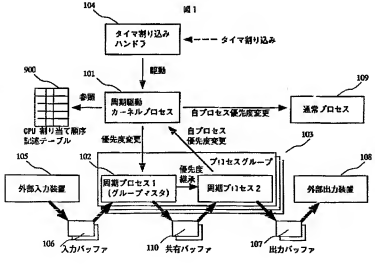
【0195】さらに、本発明の CPU 割り当てアルゴリズムでは、各プロセスグループに割り当てられる時間をできるだけ連続にとる。そのため、プロセススイッチ回数が最小限に押さえられる。プロセススイッチに要するオーバーヘッド削減による連続メディア処理の性能向上も期待できる。

【0196】さらに、本発明の CPU 割り当てアルゴリズムでは、各プロセスグループ間の実行時間比は常に一定に保たれる。そのため、ランデブなどの同期機構を使用しなくとも、ストリーム間同期の実現が可能になる。

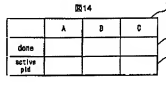
【0197】さらに本発明は、非同期イベントが発生した際の連続メディア処理を行うプロセスの実行間隔の変動を防止できる。

【図面の簡単な説明】
【図１】本発明のスケジューリング方法におけるプロセス起動及びデータの流れを示す図である。
【図２】プロセスグループを管理するデータの構造を示す図である。
【図３】create_proc_group 関数のフローチャートである。
【図４】destroy_proc_group 関数のフローチャートである。
【図５】プロセスグループに割り当てる CPU 時間の重なりを解消する方法を示す図である。
【図６】タイムスロットテーブル作成のフローチャートである。
【図７】タイムスロットテーブルの作成例の結果を示す図である。
【図８】タイムスロットテーブルの作成例のための入力データを示す図である。
【図９】CPU 割り当て順序記述テーブルの構成を示す図である。
【図１０】プロセスを管理するデータの構造を示す図である。
【図１１】proc_raise 関数のフローチャートである。
【図１２】タイマ割り込みハンドラのフローチャートである。
【図１３】周期駆動カーネルプロセスのフローチャートである。
【図１４】実行状態記述テーブルの構成を示す図である。
【図１５】連続メディア処理プロセスの起動の流れを示す図である。
【図１６】グループマスタプロセスのプログラムを示す図である。
【図１７】スレーブプロセスのプログラムを示す図である。
【図１８】本発明のスケジューリング方法におけるプロセス起動及びデータの流れを示す図である。
【図１９】コマンドリストの構成を示す図である。
【図２０】スケジューラのフローチャートである。
【図２１】ネットワーク・パケット受信システムの構成を示す図である。
【図２２】第1レベル割り込みハンドラのフローチャートである。
【符号の説明】
１０１：周期駆動カーネルプロセス、１０２：周期プロセス、１０３：プロセスグループ、７００：タイムスロットテーブル、９００：CPU 割り当て順序記述テーブル、１４００：実行状態記述テーブル、１８０１：スケジューラ、

【図１】

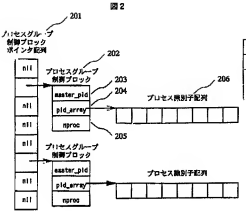


【図１４】



【図2】

図2



【図8】

図8

	801 Interval 値	802 Length 値
プロセスグループA	8	2
プロセスグループB	16	3
プロセスグループC	32	7

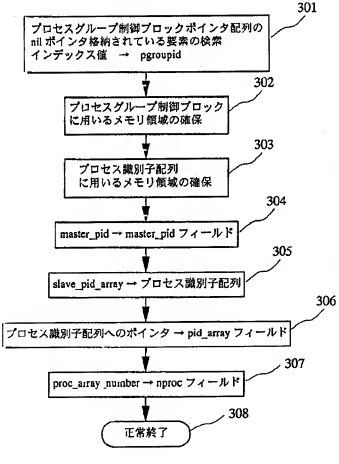
【図17】

図17

```
1701  for (i=0;i<MYTIMES;i++){
1702        /* ～周期分の記録メディア処理 */
1703        proc_raise_handoff(mastpid, PRIORITY_DEPRESSED);
1704        (proc_raise_cancel(MYSELF, PRIORITY_DEPRESSED));
1705  }
1706  wait(0);
```

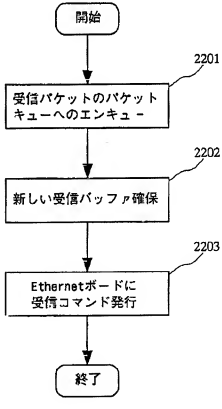
【図3】

図3



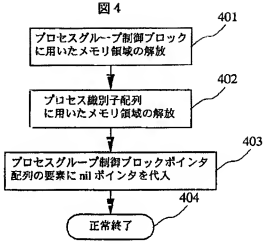
【図22】

図22



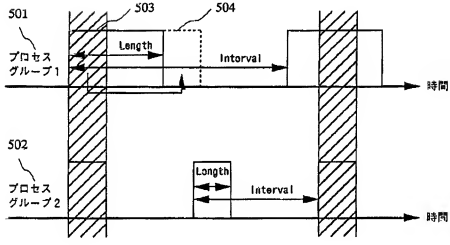
【図4】

図4



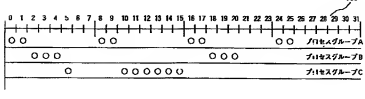
【図5】

図5



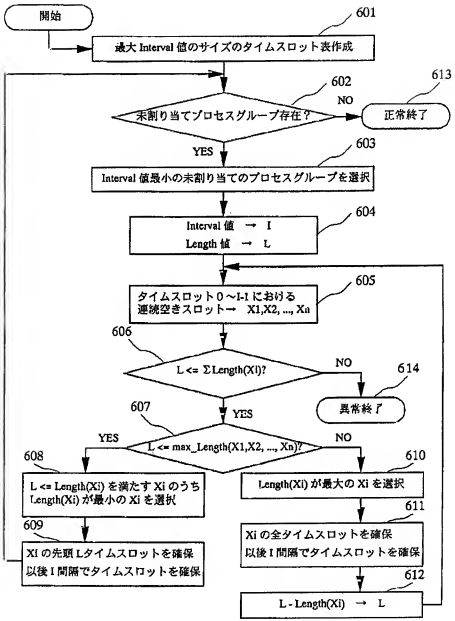
【図7】

図7



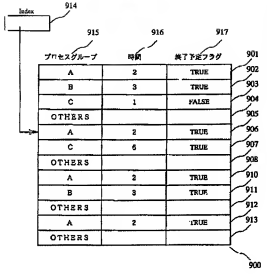
【図6】

図6


開始
601
最大 Interval 値のサイズのタイムスロット表作成
602
未割り当てプロセスグループ存在？
NO
613
正常終了
YES
603
Interval 値最小の未割り当てのプロセスグループを選択
Interval 値 → I
Length 値 → L
604
タイムスロット 0 ～I-1 における
連続空きスロット→ X1,X2, ..., Xn
605
606
L <= ΣLength(Xi)?
NO
614
異常終了
607
YES
YES
L <= max_Length(X1,X2, ..., Xn)?
NO
608
L <= Length(Xi) を満たす Xi のうち
Length(Xi) が最小の Xi を選択
610
Length(Xi) が最大の Xi を選択
611
609
Xi の先頭 Lタイムスロットを確保
以後 I 間隔でタイムスロットを確保
Xi の全タイムスロットを確保
以後 I 間隔でタイムスロットを確保
612
L - Length(Xi) → L

【図９】

図９



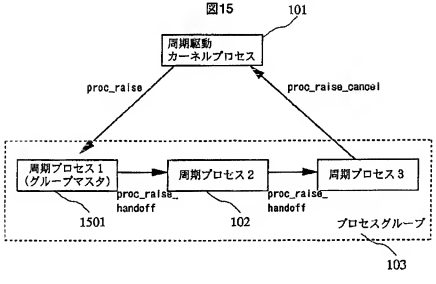
【図１６】

図16

```
1601  create_proc_group(MYSELF, slave_pid_array, proc_array_number, &pgroupid);
1602  alloc_time_slot(pgroupid, interval, length);
1603  for(i=0;i<ONTHESE199;){
1604          /* 一周期分の記録メディア処理 */
1605          proc_raise_handoff(nextpid, PRIORITY_DEPRESSED);
1606  }
1607  dealloc_time_slot(pgroupid);
1608  destroy_proc_group(pgroupid);
1609  exit(0);
```

【図１５】

図15



【図１０】

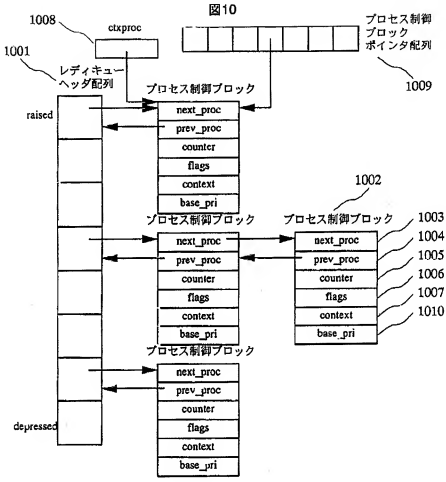


図10
1008
ctxproc
プロセス制御
ブロック
ポインタ配列
1001
レディキュー
ヘッダ配列
1009
プロセス制御ブロック
raised
next_proc
prev_proc
counter
flags
context
base_pri
1002
プロセス制御ブロック
プロセス制御ブロック
1003
1004
1005
1006
1007
1010
プロセス制御ブロック
depressed

【図１９】

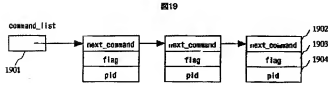
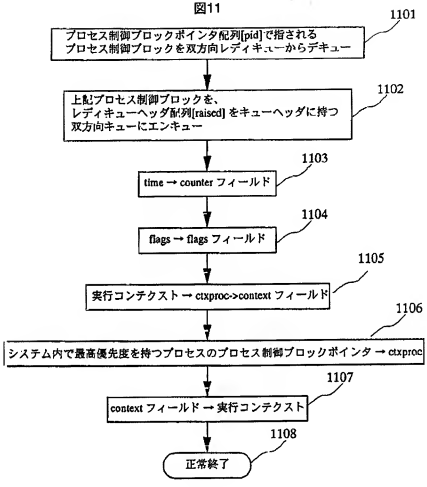


図19
command_list
next_command
flag
pid
1901
1902
1903
1904

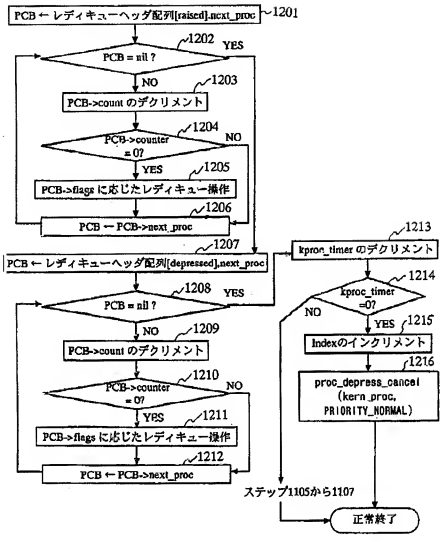
【図11】

図11


1101
プロセス制御ブロックポインタ配列[pid]で指される
プロセス制御ブロックを双方向レディキューからデキュー
1102
上記プロセス制御ブロックを、
レディキューヘッダ配列[raised]をキューヘッダに持つ
双方向キューにエンキュー
1103
time → counter フィールド
1104
flags → flags フィールド
1105
実行コンテクスト → ctxproc->context フィールド
1106
システム内で最高優先度を持つプロセスのプロセス制御ブロックポインタ → ctxproc
1107
context フィールド → 実行コンテクスト
1108
正常終了

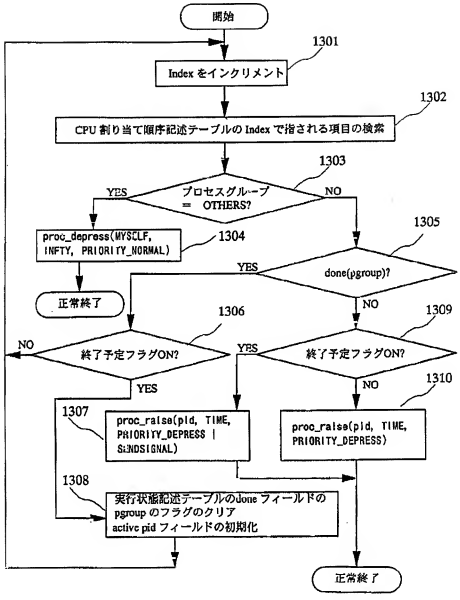
【図12】

図12


PCB ← レディキューヘッダ配列[raised].next_proc
1201
1202
YES
PCB = nil ?
NO
1203
PCB->count のデクリメント
1204
PCB->counter = 0?
NO
YES
1205
PCB->flags に応じたレディキュー操作
1206
PCB ← PCB->next_proc
1207
PCB ← レディキューヘッダ配列[depressed].next_proc
1208
YES
PCB = nil ?
NO
1209
PCB->count のデクリメント
1210
PCB->counter = 0?
NO
YES
1211
PCB->flags に応じたレディキュー操作
1212
PCB ← PCB->next_proc
1213
kpron_timer のデクリメント
1214
kproc_timer =0?
NO
YES
1215
Indexのインクリメント
1216
proc_depress_cancel (kern_proc, PRIORITY_NORMAL)
ステップ1105から1107
正常終了

【図13】

図13


開始
1301
Index をインクリメント
1302
CPU 割り当て順序記述テーブルの Index で指される項目の検索
1303
プロセスグループ == OTHERS?
YES
NO
1304
proc_deprass(MYSELF, INFTY, PRIORITY_NORMAL)
1305
done(pgroup)?
YES
NO
正常終了
1306
終了予定フラグON?
NO
YES
1309
終了予定フラグON?
YES
NO
1307
proc_raise(pid, TIME, PRIORITY_DEPRESS | SiNDSIGNAL)
1310
proc_raise(pid, TIME, PRIORITY_DEPRESS)
1308
実行状態記述テーブルのDone フィールドの pgroup のフラグのクリア
active pid フィールドの初期化
正常終了

【図18】

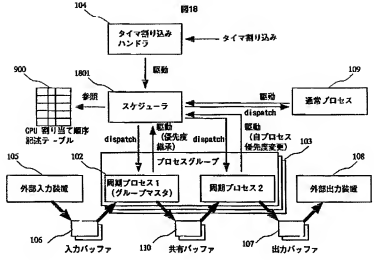


図18
104
タイマ割り込みハンドラ
タイマ割り込み
駆動
900
1801
109
参照
スケジューラ
駆動
通常プロセス
dispatch
GPU 割り当て順序記述テーブル
dispatch
駆動（優先度 dispatch 最高）
駆動（他プロセス優先度変更）
103
プロセスグループ
105
102
108
外部入力装置
同期プロセス1（グループマスタ）
同期プロセス2
外部出力装置
105
入力バッファ
130
共有バッファ
107
出力バッファ

【図21】

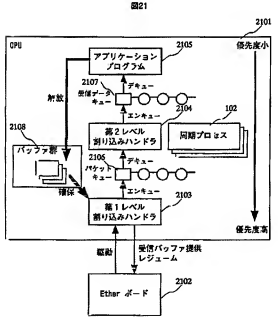
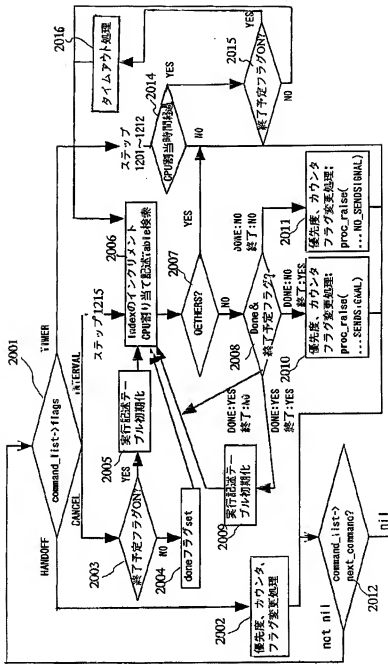


図21
2101
CPU
2105
アプリケーションプログラム
優先度小
2107
受信データキュー
デキュー
解放
エンキュー
2104
102
2108
第2レベル割り込みハンドラ
同期プロセス
バッファ群
2106
パケットキュー
デキュー
確保
エンキュー
2103
第1レベル割り込みハンドラ
優先度高
駆動
受信バッファ提供レジューム
Ether ボード
2102

【図20】



フロントページの続き

(72)発明者　岩崎　正明
神奈川県川崎市麻生区王禅寺1099番地　株
式会社日立製作所システム開発研究所内